PROJECT
TACT

CONTRIBUTIONS TO MECHANICAL MATHEMATICS

BY

JOSE RENATO ITURRIAGA-Y-DE-LA-FUENTE

april 25, 1967

C O N T R I B U T I O N S T O
M E C H A N I C A L M A T H E M A T I C S

BY

JOSE RENATO ITURRIAGA-Y-DE-LA-FUENTE

SUBMITTED TO THE CARNEGIE INSTITUTE OF TECHNOLOGY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

PITTSBURGH, PENNSYLVANIA

1967

ABSTRACT

WE DESCRIBE THE EXPERIENCE RELATING TO THE DESIGN OF A PROGRAMMING LANGUAGE, FORMULA ALGOL, WHICH IS SUITABLE FOR DESCRIBING ALGORITHMS THAT MECHANIZE SOME MATHEMATICAL TASKS. WE DEVELOP A THEORY THAT YIELDS RESULTS WITH WHICH WE CAN PROVE THE TERMINATION OF SOME MARKOV ALGORITHMS THAT PERFORM ALGEBRAIC TRANSFORMATIONS. USING THESE RESULTS WE ALSO CAN CHARACTERIZE THE CLASSES OF FORMULAE THAT CONSTITUTE THE INPUT AND OUTPUT FOR SUCH ALGORITHMS. NEXT, WE DISCUSS IN SOME DETAIL THE VARIOUS PROBLEMS RELATED TO THE MECHANIZATION OF LIMITING PROCESSES AND INEQUALITIES. TWENTY-FIVE APPENDICES --WITH RUNNING PROGRAMS-- ILLUSTRATE THE VARIOUS POINTS MADE DURING OUR PRESENTATION.

ACKNOWLEDGEMENTS

I WANT TO EXPRESS MY DEEPEST GRATITUDE TO PROFFESOR ALAN J. PERLIS, WHO NOT ONLY PROVIDED ME WITH AN ENDLESS STREAM OF IDEAS AND SUGGESTIONS FOR THIS DISSERTATION, BUT WHO ALSO HELPED ME IN MANY OTHER WAYS DURING THE YEARS I WAS AT THE CARNEGIE INSTITUTE OF TECHNOLOGY.

PETE FREEMAN AND TIM STANDISH GAVE ME INVALUABLE ASSISTANCE IN THE ARDUOUS TASK OF PRESENTING MY RESULTS IN A READABLE FORM.

I WANT TO THANK MY WIFE, ESTHER, FOR THE MORAL SUPPORT AND UNDERSTANDING WHICH SHE PROVIDED DURING THE TIME WE LIVED IN PITTSBURGH.

WHILE I WAS DOING MY GRADUATE WORK I RECIVED PARTIAL SUPPORT FROM THE CENTRO DE CALCULO ELECTRONICO OF THE UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO AND FROM THE INSTITUTO NACIONAL DE LA INVESTIGACION CIENTIFICA.

THE WORK REPORTED HERE WAS SUPPORTED BY THE ADVANCED RESEARCH PROJECT AGENCY OF THE OFFICE OF THE SECRETARY OF DEFENSE: CONTRACT SD-146.

TABLE OF CONTENTS

CHAPTER I, INTRODUCTION 1

CHAPTER II, ON FORMULA ALGOL 9

DATA STRUCTURES 13
EVALUATION AND SUBSTITUTION 23
PATTERNS 30
FORMULA PREDICATES 34
EXTRACTORS 35
MARKOV ALGORITHMS 41
SOME EXAMPLES AND FINAL REMARKS 45

CHAPTER III, ON PROOFS OF ALGORITHMS 51

STATEMENT 1 55
STATEMENT 2 59
THEOREM I 70
STATEMENT 3 78
THEOREM II 81
THEOREM III 83
STATEMENT 4 83

CHAPTER IV, ON LIMITING PROCESSES 88

TRIVIAL CASES 88
TABLE FOR DEFINING AN ALGEBRA OF INFINITY 91
HEURISTIC I 94
L' HOSPITAL'S RULE 95
HEURISTIC II 95
HEURISTIC III 95
ASYMPTOTIC ANALYSIS 97
HEURISTIC IV 98
HEURISTIC V 98
ALGORITHM FOR COMPUTING THE 'ORDER' OF A FORMULA 99
THE PROBLEM OF KNOWING THE SIGN OF '0' 101
FIRST METHOD, ASYMPTOTIC ANALYSIS 101
SECOND METHOD, INEQUALITY ANALYSIS 102
THIRD METHOD, CALCULUS ANALYSIS 103
FOURTH METHOD, ALGEBRAIC REMOVAL OF THE DIFFICULTY 104
FIFTH METHOD, RECURSIVE SEARCH 105
THE BOUNDEDNESS TECHNIQUE 105
SEQUENCES AND 'INTEGERNESS' ANALYSIS 108
ORGANIZATION OF THE PROGRAM 109
THE MAIN PROGRAM 112
THE RESTRICTED LIMIT ROUTINE 113
THE AUXILIARY LIMIT ROUTINE 114
THE EVALUATOR 117

PERFORMANCE OF THE PROGRAM 119

CHAPTER V, ON INEQUALITIES AND SIMPLIFICATION 121

INEQUALITIES 122
PROVING FOR SUFFICIENTLY LARGE N (SMALL X) 124
INDUCTION PROOFS 124
PROOF BY CASES 125
MONOTONICITY PROOFS 125
METHOD OF PROOF BY ANALYSIS 126
METHOD OF PROOF BY LOGICAL CHAIN 131
METHOD OF PROOF BY TRANSITIVITY 134
METHOD OF PROOF BY SYNTHESIS 136
SIMPLIFICATION ALGORITHMS 140

CHAPTER VI, CONCLUSIONS 145

APPENDIX A-1, THE REPLACEMENT THEOREM 152

APPENDIX A-2, BNF DEFINITIONS OF SOME CLASSES OF FORMULAE 157

BIBLIOGRAPHY 158

LIST OF APPENDICES

1. TWO PROCEDURES TO INVERT MATRICES, THE FIRST OPERATES WITH NUMBERS, THE SECOND WITH FORMULAE.

2. AN ITERATIVE DIFFERENTIATION ALGORITHM.

3. AN ITERATIVE ALGORITHM FOR SOLVING AN EQUATION ON X.

4. A RECURSIVE ALGORITHM FOR SOLVING AN EQUATION ON X.

5. A MARKOV ALGORITHM FOR SOLVING AN EQUATION ON X.

6. A MARKOV ALGORITHM FOR TRANSFORMING LOGICAL FORMULAE INTO DISJUNCTIVE NORMAL FORM.

7. A PROGRAM TO SOLVE A DIFFERENTIAL EQUATION BY COMPUTING THE FIRST N TERMS OF THE TAYLOR SERIES EXPANSION.

8. A MARKOV ALGORITHM TO ELIMINATE UNARY MINUS.

9. A MARKOV ALGORITHM TO CLEAR RATIONAL FRACTIONS.

10. A MARKOV ALGORITHM TO EXPAND A FORMULA BY MULTIPLYING OUT FACTORS.

11. A PROCEDURE TO COLLECT ALIKE TERMS IN A POLYNOMIAL AND AUXILIARY PROCEDURES.

12. A PROCEDURE FOR FACTORING OUT CONSTANTS AND VARIABLES AS WELL AS FOR COMPLETING SQUARES.

13. TWO MARKOV SCHEMATA FOR GENERATING EQUIVALENT RELATIONS IN INEQUALITY PROOFS BY LOGICAL CHAIN.

14. PROCEDURES RELATED TO LIMITING PROCESSES.

15. A PROCEDURE TO TEST FOR INTEGER VALUED FUNCTIONS.

16. A PROCEDURE TO DETERMINE THE SIGN OF A FORMULA.

17. A PROCEDURE TO ACCOMPLISH ASYMPTOTIC ANALYSIS.

18. A PROCEDURE AND MARKOV ALGORITHMS FOR CHECKING BOUNDEDNESS.

19. A PROCEDURE TO PROVE INEQUALITIES BY THE METHOD OF TRANSITIVITY.

20. ALGORITHMS FOR GENERATING CONJECTURES ABOUT INEQUALITIES.

21. A PROCEDURE TO PROVE INEQUALITIES BY THE METHOD OF ANALYSIS.

22. A PROCEDURE TO PROVE INEQUALITIES BY THE METHOD OF LOGICAL CHAIN.

23. SOME OUTPUT OF PROOFS OF INEQUALITIES.

24. SOME OUTPUT OF LIMITING PROCESSES.

25. ASYMPTOTIC ANALYSIS EXAMPLE.

CHAPTER I. INTRODUCTION

AMONG ALL THE PROBLEMS IN THE COMPUTER SCIENCES FIELD THERE IS ONE WITH CERTAIN APPEAL: THE CREATION OF THE MECHANICAL MATHEMATICIAN, OR IF THE READER WISHES, A MECHANIZED MATHEMATICAL ASSISTANT. THE PURPOSE OF THIS DISSERTATION IS TO MAKE SOME CONTRIBUTIONS TOWARD THE ACHIEVEMENT OF THIS ULTIMATE GOAL.

IT IS WELL KNOWN THAT A DIGITAL COMPUTER IS MUCH MORE THAN A VERY FAST ARITHMETIC DEVICE. IT IS INDEED A GENERAL PROCESSOR OF INFORMATION. BUT ALTHOUGH COMPUTERS HAVE BEEN A TREMENDOUS AID IN ACCOMPLISHING NUMERICAL COMPUTATIONS, THEY ARE STILL FAR FROM BEING A DAILY TOOL IN SYMBOLIC COMPUTATIONS. ADMITTEDLY THERE HAVE BEEN SPECTACULAR ACHIEVEMENTS (RUNNING PROGRAMS) THAT HAVE SHOWN THE POSSIBILITY OF A COMPUTER DOING SYMBOLIC COMPUTATIONS, BUT VERY FEW OF THESE HAVE BEEN USED IN A PRODUCTIVE WAY. IN PART THIS IS DUE TO COMPUTER LIMITATIONS, WHICH WILL BE AMELIORATED WITH THIS LATEST GENERATION OF COMPUTERS.

THE MECHANICAL MATHEMATICIAN IS ONE PROGRAM WHICH WE WOULD LIKE TO SEE UTILIZED BY MANY RESEARCHERS IN THEIR DAILY WORK. LET US MENTION SOME OF THE FEATURES, THAT OFTEN HAVE BEEN SUGGESTED, FOR THIS PROGRAM TO EMBRACE.

FOR ONE THING, IT IS EXPECTED THAT THIS PROGRAM SHOULD BE ABLE TO MANIPULATE ALGEBRAIC SYMBOLS FOLLOWING THE RULES OF A GIVEN ALGEBRA.

THESE MANIPULATIONS COULD, AT THE USER'S REQUEST, BE GOVERNED BY A LOGICAL CORE OF PROGRAMMING WHICH WILL MAKE DECISIONS AT DIFFERENT CRUCIAL POINTS. THE PROGRAM SHOULD BE ABLE TO PLAN AN ATTACK ON A GIVEN PROBLEM, AND IN SO DOING IT MIGHT OR MIGHT NOT REQUEST THE USER'S HELP. SUCH A PLAN MAY CONTAIN THE GENERATION OF SUB-PROBLEMS WHOSE SOLUTIONS ARE STEPS TOWARD THE ACCOMPLISHMENT OF THE MAIN PROBLEM. THE PLAN ALSO MAY CONTAIN VARIOUS CRITERIA TO PREVENT EXECUTION OF INVESTIGATIONS WHICH ARE EITHER VERY DIFFICULT OR IMPOSSIBLE TO ACHIEVE. THE PROGRAM MUST BE ABLE TO OBTAIN RESULTS BY PRODUCING CONSEQUENCES FROM KNOWN FACTS. IT MUST BE ABLE TO RECOGNIZE CLASSES OF PROBLEMS, AS WELL AS DEFINE THEM, SO IT CAN REALIZE WHEN A GIVEN PROBLEM IS SIMILAR TO ONE ALREADY SOLVED, OR TO ONE IT KNOWS HOW TO SOLVE. THUS INSTEAD OF ATTEMPTING TO GENERATE A SOLUTION BY STARTING FROM SCRATCH IT WILL TRY TO FIND A BRIDGE BETWEEN THE PROBLEM AND THE KNOWN FACTS. TO LAY DOWN SUCH A BRIDGE MEANS THAT THE PROGRAM MUST FIND A SEQUENCE OF TRANSFORMATIONS THAT WILL MAP THE ORIGINAL PROBLEM INTO AN EQUIVALENT AND --ACCORDING TO ITS CAPABILITIES-- A SOLVABLE ONE. THE PROGRAM MUST BE ABLE TO MODIFY AND DIRECT ITS APPROACHES AS A CONSEQUENCE OF ITS PARTIAL SUCCESSES OR FAILURES. FINALLY THE PROGRAM MUST BE ABLE TO UNDERSTAND AND SPEAK A SUBSET OF A NATURAL LANGUAGE IN WHICH A DIALOGUE CAN BE PERFORMED BETWEEN MACHINE AND USER ABOUT THE STATUS OF THE COMPUTATION.

THERE ARE MANY REASONS WHY RESEARCHERS HAVE BEEN INTERESTED IN THE MECHANICAL MATHEMATICIAN. FOR SOME, THIS PROGRAM WOULD BE A MODEL TO SUPPORT A PSYCHOLOGICAL THEORY ABOUT THE BEHAVIOR OF A HUMAN BRAIN DOING

MATHEMATICS. FOR OTHERS, THIS PROGRAM WOULD BE A TEACHING ASSISTANT, AN ACTIVE AGENT WHICH WILL GIVE ANSWERS, ADVICE AND CRITICISM TO THE STUDENT OF MATHEMATICS WHILE DOING HOMEWORK. FOR OTHERS, IT WOULD BE A TRUE RESEARCH ASSISTANT WHICH WILL DO MATHEMATICAL DATA PROCESSING FOR ITS MASTER, THE HUMAN SCIENTIST. AND YET FOR OTHERS, THE MECHANICAL MATHEMATICIAN WOULD BE A FORMAL OBJECT BY ITSELF: TO BE A SUBJECT OF STUDY IN THE COMPUTER SCIENCES.

WE WANT A COMPUTER TO DO MATHEMATICS. BUT COMPUTERS CAN DO ONLY THOSE THINGS THAT CAN BE DESCRIBED IN A FORMAL AND PRECISE WAY, I.E., ALGORITHMICALLY. TO MECHANIZE MATHEMATICS WE NEED A COLLECTION OF WELL DEFINED RULES TO SOLVE PROBLEMS IN THIS AREA. MATHEMATICS, THE MOST FORMAL INTELLECTUAL ACTIVITY OF ALL, HAS AN ASTONISHINGLY LARGE COLLECTION OF NON-FORMAL RULES TO CHOOSE TECHNIQUES FOR SOLVING PROBLEMS. THE POSSESSION AND ARTFUL USE OF THESE RULES IS WHAT WE MIGHT CALL IN HUMANS 'MATHEMATICAL MATURITY' --A SUBTLE ABILITY THAT ALLOWS ONE TO FIND OR INVOKE THE RIGHT PIECE OF KNOWLEDGE WHICH WILL LEAD TO THE SOLUTION OF THE PROBLEM. IN MOST HUMANS, MATHEMATICAL MATURITY IS OBTAIN BY EXPERIENCE. BUT IT IS NOT CLEAR HOW ONE WOULD OR EVEN COULD MECHANIZE MATHEMATICAL MATURITY. THERE IS AN INCREASING NEED FOR A NEW BRANCH OF MATHEMATICS WHOSE MAIN GOAL IS THE STUDY OF MATHEMATICAL PROBLEM SOLVING. THIS BRANCH HAS BEEN CALLED MECHANICAL MATHEMATICS [35].

IN OUR RESEARCH WE TRIED TO OBTAIN AS MUCH INSIGHT AS POSSIBLE INTO THE CHARACTERISTICS OF PROBLEM SOLVING IN SOME AREAS OF MATHEMATICAL ANALYSIS. SOME OF OUR RESULTS HOPEFULLY WILL BRING SOME

UNDERSTANDING OF THE ISSUES RELATED TO THE DEVELOPMENT OF MECHANICAL MATHEMATICS.

WHEN WE SOLVE A PROBLEM IN MATHEMATICS WE USE KNOWLEDGE AND INTELLIGENCE. THE MECHANICAL MATHEMATICIAN HAS TO HAVE BOTH. THE INTELLIGENCE OR REASONING IN A MECHANICAL DEVICE IS A SUPER-STRUCTURE THAT OPERATES ABOVE A CORE OF KNOWLEDGE. THIS IS A SIMPLISTIC BUT USEFUL WAY TO SEE THE PROBLEM. BECAUSE OF THE RECURSIVE NATURE OF MOST PROGRAMS IN THIS AREA, IT IS DIFFICULT TO SEPARATE PROGRAMMING CONTROL FROM ACTUAL COMPUTATION, I.E., WITH THE READER'S PERMISSION, REASONING FROM KNOWLEDGE. ALTHOUGH WE CANNOT DRAW A CLEAR BOUNDARY BETWEEN THE USE OF INTELLIGENCE AND THE USE OF KNOWLEDGE FOR SOLVING A PROBLEM, IN THIS DISSERTATION WE SHALL FOCUS OUR ATTENTION PRIMARILY ON THE QUESTION OF PROVIDING KNOWLEDGE TO A MECHANICAL MATHEMATICIAN.

KNOWLEDGE IS ACQUIRED BY TRAINING. IN PREPARING A MECHANICAL MATHEMATICIAN ONE CANNOT DO WITH ANY LESS THAN IN PREPARING A HUMAN MATHEMATICIAN. INDEED ONE HAS TO DO MORE, SINCE BY ALMOST ANY MEASUREMENT STANDARD, MACHINES ARE MUCH LESS INTELLIGENT THAN HUMANS. THE ONLY ADVANTAGE THE MACHINE WILL HAVE OVER THE HUMAN IS THE PRECISION OF ITS ACTIONS AND THE DEPTH OF RECURSION AND BRANCHING IT CAN MANAGE IN ITS CONTROL. IN OTHER WORDS, THE MACHINE CAN ACCOMPLISH A BIGGER HEURISTIC SEARCH. FURTHERMORE IN THE MECHANIZATION OF PROBLEM SOLVING, SOME OTHER PROBLEMS APPEAR FOR THE FIRST TIME. FOR EXAMPLE, PROVING TERMINATION OF AN ALGORITHM MAY NOT BE CONCERNED WITH MATHEMATICS, BUT WITH META-MATHEMATICS, E.G., SOME FORMAL SYSTEM IN WHOSE TERMS PROOFS ARE ENGINEERED.

BEFORE WE CAN HAVE A MATHEMATICAL ASSISTANT WE NEED AN EDUCATED ANALYST. WE EDUCATE PEOPLE BY TEACHING THINGS WE THINK MAY BE USEFUL TO THEM. WE SHALL EDUCATE MACHINES BY PROGRAMMING TOGETHER ALGORITHMS AND TECHNIQUES WE THINK COVER THE VARIOUS AREAS ON WHICH THEY ARE SUPPOSED TO OPERATE. WE SHALL PROVIDE THE MACHINE WITH TOOLS OF THE SAME KIND WE GIVE TO STUDENTS TO SOLVE PROBLEMS. FOR EXAMPLE, WHEN WE TRAIN THEM TO DO LIMITING PROCESSES WE TEACH THEM HOW TO USE L'HOSPITAL'S RULE.

THE DESIGN OF AN APPROPRIATE LANGUAGE TO DESCRIBE ALGORITHMS FOR THE MECHANICAL MATHEMATICIAN IS AN IMPORTANT STEP TOWARD ITS ACHIEVEMENT. ONCE A LANGUAGE HAS BEEN DESIGNED, WE TRAIN THE COMPUTER TO UNDERSTAND SUCH A LANGUAGE AND THEN WE USE IT TO TRAIN THE COMPUTER IN APPLYING TECHNIQUES AND ALGORITHMS. THE ORDER IN WHICH WE PRESENT VARIOUS TECHNIQUES CANNOT BE ARBITRARY. THERE IS A PARTIAL ORDER BETWEEN PROBLEMS OR TASKS CONNECTED WITH THE MECHANICAL MATHEMATICIAN. BEFORE THERE IS ANY HOPE TO SOLVE SOME CLASS OF PROBLEMS, IT IS MANDATORY THAT WE UNDESTAND HOW TO SOLVE MORE BASIC PROBLEMS. FOR EXAMPLE, BEFORE WE ATTEMPT TO MECHANIZE THE TESTING FOR CONVERGENCE OF A SERIES, WE SHOULD HAVE SOLVED THE MECHANIZATION PROBLEM FOR LIMITING PROCESSES, THE MANIPULATION OF INEQUALITIES, AND THE TRANSFORMATION OF ALGEBRAIC EXPRESSIONS.

THE FIRST GOAL OF THIS DISSERTATION IS TO OBTAIN A LANGUAGE WHICH WILL ENABLE A COMPUTER TO ACCEPT NATURAL INSTRUCTIONS FOR THE TASKS OF ALGEBRA AND MATHEMATICAL ANALYSIS.

A SECOND PARALLEL GOAL OF THIS DISSERTATION IS THE DISCOVERY OF

ALGORITHMS WHICH ARE SUITABLE FOR COMPUTERS. IN OUR SEARCH FOR ALGORITHMS WE REQUIRED THAT THEIR DESCRIPTION IN THE LANGUAGE SHOULD REVEAL CLEARLY THEIR STRUCTURE. THIS REQUIREMENT HAD A VERY USEFUL INFLUENCE ON THE DESIGN OF THE LANGUAGE.

A THIRD GOAL OF THIS DISSERTATION IS THE STUDY OF THE ALGORITHMS THEMSELVES. THUS WE SHALL CONSIDER QUESTIONS RELATED TO THE TERMINATION OF ALGORITHMS AND WITH THE FORMAL DESCRIPTION OF THEIR INPUTS AND OUTPUTS. ALSO WE EXAMINE THE MATHEMATICAL VALIDITY OF SOME ALGORITHMS.

IN CHAPTER II WE DESCRIBE OUR EXPERIENCE IN THE DESIGN AND IMPLEMENTATION OF A LANGUAGE, CALLED FORMULA ALGOL, WHICH WE THINK IS USEFUL FOR DESCRIBING ALGORITHMS FOR THE MECHANICAL MATHEMATICIAN. IN CHAPTER III WE DEVELOP A THEORY THAT YIELDS RESULTS WITH WHICH WE CAN PROVE THE TERMINATION OF SOME ALGORITHMS FOR ALGEBRAIC MANIPULATION. THOSE RESULTS ALSO ALLOW US TO CHARACTERIZE THE CLASS OF FORMULAE FOR WHICH AN ALGORITHM WILL OPERATE AS WELL AS THE CLASS OF FORMULAE CHARACTERIZED BY THE ALGORITHM'S OUTPUT.

IN CHAPTERS IV AND V WE EXPLORE NEW AREAS OF ALGEBRAIC MANIPULATION. NOT ONLY TO PRESENT AN ORIGINAL COLLECTION OF ALGORITHMS THAT SHOW THE USEFULNESS OF FORMULA ALGOL, BUT ALSO TO INVESTIGATE THE POSSIBILITY OF MECHANIZING AREAS OF MATHEMATICS NOT YET CONSIDERED BY THE COMPUTER SCIENTIST. BY DOING THIS, WE EXPECT BOTH TO BRING NEW ISSUES TO THE SYMBOL MANIPULATION ARENA AND TO PRESENT NEW EVIDENCE TO SUPPORT VIEWS ALREADY EXPRESED IN THE LITERATURE.

THE PROBLEMS WE HAVE CHOSEN COME FROM MATHEMATICAL ANALYSIS. THE FIRST CONTACT THAT A FRESHMAN STUDENT HAS WITH ANALYSIS IS INVARIABLY THE CONCEPT OF LIMIT. THUS LIMITING PROCESSES PROVIDED US A SOURCE OF CONCRETE PROBLEMS TO WORK WITH. THE MECHANIZATION OF NEW TASKS GAVE US THE OPPORTUNITY TO GATHER TOGETHER SEVERAL KNOWN TECHNIQUES AND UTILIZE THEM TO PRODUCE BUILDING BLOCKS FOR VARIOUS ALGORITHMS. IN CHAPTER IV WE SHALL DISCUSS LIMITING PROCESSES IN GENERAL. IN CHAPTER V WE SHALL DISCUSS THE TASKS WE HAVE PROGRAMMED TO PROVIDE ALGEBRAIC MANIPULATION SUPPORT FOR THE LIMITING PROCESSES.

WHEN WE WERE ALREADY WORKING ON THIS PROBLEM R.R. FENICHEL PUBLISHED [12] SOME INTERESTING RESULTS RELATED TO COMPUTATION OF LIMITS. HE POINTS OUT THAT, AS A DIRECT CONSEQUENCE OF RICHARDSON'S RESULT [29],[37] THERE IS NO RECURSIVE DECISION PROCEDURE FOR LIMITS OF FUNCTIONS COMPOSED WITH THE RATIONALS, PI, LOG 2, ADDITION, MULTIPLICATION, DIVISION, SIN, EXP, AND ABSOLUTE VALUE. BUT THIS NEGATIVE RESULT SEEMS TO DISCOURAGE HIM IN HIS EFFORT TO STUDY THE MECHANIZATION OF LIMITS. HE DESCRIBES A COLLECTION OF RULES TO COMPUTE ONLY DOUBLE-SIDED LIMITS OF PIECEWISE ANALYTIC FUNCTIONS AND HE USES L'HOSPITAL'S RULE AS THE ONLY EXTRA TOOL TO SUPPLEMENT WHAT WE ARE GOING TO CALL TRIVIAL LIMITS.

OUR APPROACH TO LIMITING PROCESSES IS ORGANIZED IN A COMPLETELY DIFFERENT MANNER THAN FENICHEL'S. NONETHELESS, OUR PROGRAM INCLUDES ALL THE CAPABILITIES THAT HE MENTIONS AS PRESENT IN HIS PROGRAM AND FURTHERMORE WE HAVE EXPLORED NEW RELATED AREAS THAT HAVE NOT BEEN CONSIDERED BEFORE IN THE LITERATURE. SUCH AREAS ARE:

1.- ONE-SIDE LIMITS OR RESTRICTED LIMITS.

2.- A COMPLETE ALGEBRA OF INFINITY.

3.- ASYMPTOTIC BEHAVIOR ANALYSIS.

4.- BOUNDEDNESS ANALYSIS.

5.- DETECTION OF INTEGER VALUED FUNCTIONS.

6.- DETECTION AND PROOF OF INEQUALITIES (SIGN ANALYSIS).

7.- MECHANIZATION OF THE PROCESSES: ''FOR SUFFICIENTLY SMALL X, PROVE F(X)>0'' OR ''FOR SUFFICIENTLY LARGE N,PROVE G(N)>0''.

8.- ANALYSIS OF THE SIGN OF '0' OR, EQUIVALENT, ANALYSIS OF THE SIDE FROM WHICH A LIMIT VALUE IS OBTAINED.

IN CHAPTER VI WE SUMMARIZE OUR FINDINGS AND CRITICIZE OUR WORK.

CHAPTER II, ON FORMULA ALGOL

THE FORMULA ALGOL [28],[16] PROJECT HAS BEEN A COLLECTIVE ENTERPRISE* EXTENDING OVER THE PAST FOUR YEARS. WITHIN THIS PROJECT THE AUTHOR'S MAIN EFFORTS HAVE BEEN DEVOTED TO DEFINING, EXPLORING, AND IMPROVING THE FORMULA MANIPULATION PART OF THE LANGUAGE. IN THIS CHAPTER WE DESCRIBE SOME EXPERIENCE RELATING TO THE DEVELOPMENT OF THE FORMULA MANIPULATION FEATURES, AND HOPE TO RECORD HERE VARIOUS REVEA_ING OBSERVATIONS RESULTING FROM THIS LANGUAGE DESIGN EXPERIENCE.

IN PARTICULAR, WE SHALL CONSIDER BASIC TASKS OF ALGEBRAIC MANIPULATION AND OBSERVE HOW THOSE TASKS ARE DESCRIBED IN FORMULA ALGOL. WE SHALL STUDY THE EVOLUTION OF ITS SYNTAX AND SEMANTICS AS REPORTED IN [26],[27],[28] AS WELL AS INTERMEDIATE STEPS NOT RECORDED IN OTHER PUBLICATIONS. WE SHALL POINT OUT HOW OUR EXPERIENCE USING THE LANGUAGE PROVIDED FEEDBACK ON THE DESIGN, LEADING TO LANGUAGE MODIFICATIONS, EITHER BY IMPROVING ITS SYNTAX, OR BY ENLARGING THE MEANING OF ALREADY EXISTING CONSTRUCTIONS, OR BY SUGGESTING THE CREATION OF ENTIRELY NEW SYNTACTIC CONSTRUCTIONS TOGETHER WITH THEIR CORRESPONDING SEMANTIC DEFINITIONS.

---

* DR. ALAN J. PERLIS PROPOSED THIS RESEARCH AND HE HAS GUIDED AND INSPIRED THIS COLLECTIVE EFFORT. TIM STANDISH IS RESPONSIBLE FOR THE DESIGN AND IMPLEMENTATION OF THE GENERAL LIST PROCESSING CAPABILITIES. RENATO ITURRIAGA IS RESPONSIBLE FOR THE DESIGN AND IMPLEMENTATION OF THE FORMULA MANIPULATION CAPABILITIES. RUDY KRUTAR AND JAY EARLY WORKED VERY HARD ON VARIOUS IMPLEMENTATION PROBLEMS.

PERHAPS, THE BEST WAY TO START DESCRIBING FORMULA ALGOL IS TO ANSWER THE QUESTION: WHY WAS THIS LANGUAGE CREATED AT ALL? WE POINTED OUT IN CHAPTER I THAT THE DESIGN OF A SUITABLE LANGUAGE FOR ALGEBRAIC MANIPULATIONS IS A VERY IMPORTANT STEP TOWARD THE ACHIEVEMENT OF THE MATHEMATICAL ASSISTANT. IN EARLY 1963, WHEN THE FORMULA ALGOL PROJECT STARTED, THERE WERE ALREADY POWERFUL LANGUAGES LIKE IPL-V [24] OR LISP [21] WHICH HAD BEEN USED TO PRODUCE SPECTACULAR ACHIEVEMENTS -- PROVING THEOREMS ON PROPOSITIONAL CALCULUS [25] USING IPL IN 1957 OR THE FORMAL INTEGRATION OF FORMULAE [32] USING LISP IN 1961. UNFORTUNATELY THESE LANGUAGES WERE TOO BASIC. THEY WERE DESIGNED TO CONTAIN VERY FEW FUNDAMENTAL CONSTRUCTIONS FROM WHICH ANY ARTIFICIAL INTELLIGENCE PROBLEM COULD BE ATACKED; THUS IT WAS POSSIBLE TO MAKE USE OF THEM FOR A PARTICULAR APPLICATION IN MATHEMATICS, BUT THE COMPLEXITY IN THE PROGRAMMING WAS TREMENDOUS. IN FACT THE EXAMPLES MENTIONED ABOVE STILL REMAIN AS LAND MARKS OF WHAT IT HAS BEEN POSSIBLE TO ACHIEVE USING THOSE LANGUAGES IN MATHEMATICAL PROBLEMS. ONE OF THE PROBLEMS WITH SUCH 'PURE LIST PROCESSING LANGUAGES' IS THAT NUMERICAL EVALUATION PHASES ARE CUMBERSOME TO EXPRESS AND INEFFICIENT TO EXECUTE. A SOLUTION [14] PROPOSED THE ADDITION OF LIST MANIPULATION PROCEDURES TO A NUMERICALLY ORIENTED LANGUAGE (FORTRAN), BUT VERY LITTLE GAIN WAS MADE TO EASE SYMBOLIC COMPUTATIONS. STILL AN OUTSTANDING RESULT WAS PRODUCED: A PROGRAM WHICH COULD PROVE THEOREMS IN GEOMETRY [12].

CERTAINLY AN IMPROVEMENT IN SPEED AND IN MEMORY CAPACITY OF COMPUTERS WILL HELP TO INCREASE THE SCOPE OF THE ACHIEVEMENTS IN SYMBOLIC COMPUTATIONS. NEVERTHELESS, EVERYBODY IN THE COMPUTER FIELD

AGREES THAT AN IMPROVEMENT SHOULD COME IN OUR PROGRAMMING TECHNIQUES. WE NEED BOTH LANGUAGES WITH MORE COMPUTATIONAL POWER AND BETTER ALGORITHMS OPERATING ON MORE FLEXIBLE DATA STRUCTURES. THE FIRST FORMULA ALGOL LANGUAGE [26] WAS CREATED AS AN ANSWER TO THE FIRST NEED: A LANGUAGE WITH MORE COMPUTATIONAL POWER IN ALGEBRAIC MANIPULATION PROBLEMS. A SINGLE FORMULA ALGOL STATEMENT MAY BE EQUIVALENT TO SEVERAL HUNDREDS OF IPL PRIMITIVE INSTRUCTIONS OR EQUIVALENT TO A LISP S-EXPRESSION BULKY WITH NESTED PARENTHESES.

AN ADDITIONAL DESIGN OBJECTIVE OF FORMULA ALGOL RELATES TO THE FOLLOWING.

IN THE SPRING OF 1964 IT WAS RECOGNIZED THAT PROGRAMS WERE BEING WRITTEN WHICH UTILIZED SEVERAL DISTINCT VARIETIES OF SYMBOLIC COMPUTATIONS. FOR EXAMPLE, BOBROW'S STUDENT PROGRAM [5] WRITTEN IN LISP, ACCEPTED WORD PROBLEMS IN ALGEBRA STATED IN A RESTRICTED FORM OF ENGLISH; IT TRANSLATED THEM INTO SYSTEMS OF ALGEBRAIC EQUATIONS USING INFORMATION STORED IN A FACTUAL DATA BASE, AND IT SOLVED THESE SYSTEMS OF EQUATIONS TO PRODUCE SOLUTIONS TO THE PROBLEMS. TO TRANSLATE THE RESTRICTED ENGLISH INPUTS, BOBROW CONSTRUCTED AND USED A SIMULATION OF THE STRING PROGRAMMING LANGUAGE COMIT [10] WHICH HE CALLED METEOR [4]. HE USED THE SOMEWHAT PRIMITIVE ARITHMETIC COMMANDS OF LISP (NOW IMPROVED IN LISP II [1]) TO PERFORM ARITHMETIC OPERATIONS AND HE USED THE LIST PROCESSING FACILITIES OF LISP BOTH TO PERFORM FORMULA MANIPULATIONS IN THE SOLVING OF EQUATIONS AND IN THE GENERAL ADMINISTRATION OF HIS PROGRAM. THIS TYPE OF PROGRAM ILLUSTRATES THE NEED FOR A LANGUAGE WHICH COMBINES THE ADVANTAGES OF SEVERAL FORMS OF SYMBOLIC AND NUMERIC

COMPUTATION IN ONE SYSTEM.

ONE OF THE OBJECTIVES IN THE DESIGN OF LATER VERSIONS OF FORMULA ALGOL WAS TO PROVIDE A SYMBOLIC COMPUTATION FACILITY IN WHICH THE ADVANTAGES OT THESE SEVERAL TYPES OF COMPUTATIONS WERE COMBINED.

THESE FACILITIES ARE FOUND IN THE PRESENT SYSTEM, BUT THEIR SPECIFIC PROPERTIES DO NOT RELATE TO THE PRESENT DISCUSSION OF FORMULA MANIPULATION AND WE WILL NOT PURSUE THEM HERE. THE INTERESTED READER WILL FIND THEM DESCRIBED IN ''A DEFINITION OF FORMULA ALGOL'' [28]

ALGOL-60 [23] IS A POWERFUL LANGUAGE FOR NUMERIC COMPUTATIONS; IT PROVIDES GOOD CONTROL MECHANISMS; IT PERMITS RECURSIVE PROGRAMMING; IT HAS A CONVENIENT BLOCK STRUCTURE WITH DECLARATIVE AND IMPERATIVE STATEMENTS; FINALLY IT UTILIZES A MATHEMATICAL LIKE NOTATION. WE CHOSE TO EXPAND ALGOL-60 BY ADDING NEW SYNTACTIC CONSTRUCTIONS AND/OR SEMANTIC INTERPRETATIONS.

THE FORMULA MANIPULATION ADDITIONS WERE MADE FOLLOWING TWO DIALECTIC PRINCIPLES OF DESIGN.

1.- THERE ARE CERTAIN FUNDAMENTAL ACTIONS OUT OF WHICH MOST OF OUR MATHEMATICAL MANIPULATIONS ARE BUILT. WE CHOOSE SOME OF THOSE ACTIONS TO BE PRIMITIVES IN THE LANGUAGE. THESE FORMULA MANIPULATION PRIMITIVES ARE FAR FROM BEING PRIMITIVES IN THE COMPUTATIONAL SENSE, I.E., PRIMITIVES LIKE IPL'S J60 OR LISP'S 'CAR'. IT IS HERE WHERE WE GAIN COMPUTATIONAL POWER.

2.- BUT THIS GAIN IN COMPUTATIONAL POWER HAS MEANING ONLY IF ONE DOES NOT PRECLUDE THE EXISTENCE OF COMPUTATIONAL PRIMITIVES. THERE MUST BE PRIMITIVES TO CONSTRUCT, TO TEST AND TO BREAK INTO PIECES ANY DATA STRUCTURE DEFINED IN THE LANGUAGE.

FINALLY, DURING THE ACTUAL DESIGN WE TRIED TO FOLLOW ALL THE STANDARD PRINCIPLES OF DESIGN, FOR EXAMPLE, THE LANGUAGE SHOULD BE UNAMBIGUOUS, READABLE, GENERAL, CONSISTENT, ETC.

ABOUT THE SAME TIME THAT FORMULA ALGOL WAS CONCEIVED OTHER ALGEBRAIC LANGUAGES, NOTICEABLY FORMAC [31] AND ALPAK [6], WERE BEING DEVELOPED ELSEWHERE.

DATA STRUCTURES.

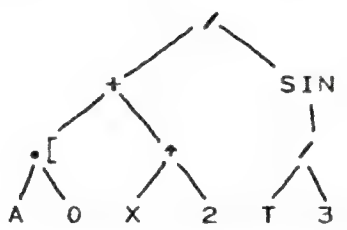
THE FIRST FACT ABOUT AN ALGEBRAIC MANIPULATION LANGUAGE THAT ONE WANTS TO KNOW IS WHAT KIND OF FORMULAE CAN BE USED, HOW SUCH FORMULAE ARE CREATED, AND HOW ARE THEY REPRESENTED INSIDE THE COMPUTER MEMORY. FORMULA ALGOL HAS THE TYPE DECLARATIONS: INTEGER, REAL AND BOOLEAN, OF ALGOL-60; BUT IT ALSO HAS A NEW TYPE: FORM. A VARIABLE OF TYPE FORM CAN HAVE A FORMULA AS VALUE. FORMULAS ARE CREATED BY WRITING DOWN A ONE-DIMENSIONAL INFIX DESCRIPTION OF THEM. FOR EXAMPLE, THE MATHEMATICAL EXPRESSION

$$\frac{A_0 + X^2}{\sin Y}$$

IS DESCRIBED IN FORMULA ALGOL BY

```
(A[0] + X↑2)/SIN(Y)
```

FURTHEMORE, IF THE VALUE OF X IS ITSELF, THE VALUE OF Y IS T/3 AND THE VALUE OF A[0] IS ITSELF, THE FOLLOWING TREE STRUCTURE WILL BE CREATED:



NOTICE THAT THE CONSTRUCTION OF FORMULAE SO FAR IS ACHIEVED WITHOUT INTRODUCTION OF NEW SYNTACTIC ELEMENTS IN THE ALGOL LANGUAGE, BUT RATHER BY EXPANDING THE MEANING OF ALREADY EXISTING CONSTRUCTIONS THROUGH THE USE OF A NEW TYPE DECLARATION. THE MEANING OF

A + B

IN THE LANGUAGE DEPENDS ON THE TYPES OF A AND B. IF BOTH TYPES ARE EITHER REAL OR INTEGER THEN A+B HAS A NUMBER AS VALUE WHICH WOULD BE AN INTEGER IF AND ONLY IF A AND B ARE OF TYPE INTEGER. ON THE OTHER HAND, IF EITHER A OR B ARE OF TYPE FORM THEN THE VALUE OF A+B IS A FORMULA.

SINCE TYPE INFORMATION GIVES THE MEANING OF AN EXPRESSION SUCH INFORMATION CAN BE USED TO COMPILE APPROPRIATE CODE. THUS NUMERIC COMPUTATIONS ARE ACHIEVED USING THE FULL ARITHMETIC POWER OF THE MACHINE EVEN IF THEY ARE MIXED WITH SYMBOLIC COMPUTATIONS WHICH ARE GENERALLY MORE EXPENSIVE TO ACHIEVE.

THE FOLLOWING EXAMPLE ILLUSTRATES HOW THE MEANING OF A PROGRAM CAN BE COMPLETLY ALTERED JUST BY CHANGING SOME OF ITS TYPE DECLARATIONS. THERE ARE ALGORITHMS WHICH DESCRIBE TECHNIQUES EQUALLY APPLICABLE AND MEANINGFUL WHEN THEY ARE USED EITHER WITH NUMBERS OR FORMULAS. CONSIDER THE INVERSION OF A MATRIX. ONE COULD TAKE THE ALGOL-60 PROGRAM, WHICH HAS

BEEN WRITTEN OBVIOUSLY FOR NUMERIC COMPUTATIONS, AND WITH A SUBSTITUTION OF SOME OF THE OCCURRENCES OF THE BASIC SYMBOLS 'REAL' AND 'INTEGER' FOR THE NEW BASIC SYMBOL 'FORM', A FORMULA ALGOL PROGRAM IS CREATED. THIS NEW PROGRAM WILL ACCEPT A MATRIX OF FORMULAE AS INPUT, TO PRODUCE ANOTHER MATRIX OF FORMULAE WHICH IS THE INVERSE. EACH ONE OF THE ELEMENTS OF THE INVERSE MATRIX COULD BE 'SIMPLIFIED' OR ALGEBRAICALLY TRANSFORMED WITH SOME CRITERIA THAT WILL MINIMIZE THE ROUND OFF ERROR GENERATED WHEN NUMERICAL VALUES ARE SUBSTITUTED FOR FORMULA VARIABLES. APPENDIX-1 HAS TWO PROGRAMS WHICH ARE IDENTICAL EXCEPT THAT TWO OCCURRENCES OF THE WORD 'REAL' IN THE FIRST PROGRAM ARE REPLACED BY THE WORD 'FORM' IN THE SECOND PROGRAM; THEY HAVE ALSO DIFFERENT INPUT. ONE PROGRAM COMPUTES THE INVERSE OF A LOWER TRIANGULAR MATRIX OF NUMBERS, THE OTHER PROGRAM COMPUTES THE INVERSE OF A LOWER TRIANGULAR MATRIX OF FORMULAE. THE LATTER PRINTS THE MATRIX ELEMENTS DIRECTLY AS COMPUTED AND AFTERWARDS SIMPLIFIES THEM.

THE FIRST VERSION OF THE LANGUAGE [26] INCLUDED ONLY THE ELEMENTARY ALGEBRAIC FORMULAE. I.E., THOSE FORMULAE WHOSE CONNECTORS ARE THE BINARY OPERATORS: '+', '-', '*', '/', '↑', '=', '<', '>', '¬=', '¬<', '¬>', '^', 'v' AND '→' TOGETHER WITH THE UNARY OPERATORS '-', 'SIN', 'COS', 'ARCTAN', 'EXP', 'LN', 'SQRT', 'ABS', 'ENTIER', 'SIGN' AND '¬'.

THE LANGUAGE'S FORMULAE COULD BE CONSIDERED AS THE REPRESENTATION OF MATHEMATICAL FUNCTIONS, BUT NOT VERY LONG AFTER WE IMPLEMENTED THE FIRST FORMULA ALGOL SYSTEM WE REALIZED THAT TWO BASIC CAPABILITIES WERE MISSING: FORMULAE DESCRIBING DISCONTINUOUS FUNCTIONS AND ARBITRARY FUNCTIONAL RELATIONS. THIS SUGGESTED THE '.IF' AND THE '.('

CONSTRUCTIONS WHICH LED TO OTHER 'DOT-CONSTRUCTIONS'.

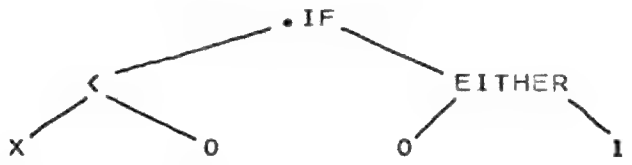
CONSIDER THE STEP FUNCTION S(X),

$$S(X) = \begin{cases} 0 \text{ IF } X<0 \\ 1 \text{ IF } X>0 \end{cases}$$

WHICH IS DESCRIBED IN FORMULA ALGOL AS

.IF X<0 THEN 0 ELSE 1

THE TREE STRUCTURE ASSOCIATED WITH THIS CONSTRUCTION IS



ON THE OTHER HAND THE VALUE OF 'IF X<0 THEN 0 ELSE 1' (WITHOUT THE DOT) IS NOT A FORMULA BUT A NUMBER WHOSE VALUE IS EITHER 0 OR 1 DEPENDING ON THE CURRENT VALUE OF X.

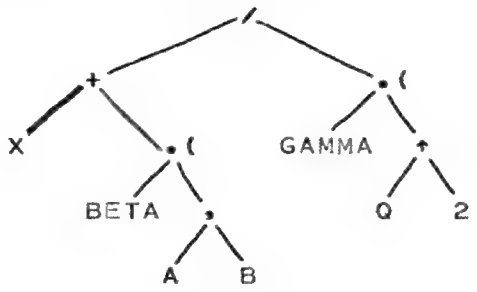
NOW, SUPPOSE WE WANT TO WORK WITH THE GAMMA AND THE BETA FUNCTIONS AS SUCH, I.E. WE ARE NOT INTERESTED IN COMPUTING NUMERICAL VALUES BUT OPERATING SYMBOLICALLY WITH THEM. THERE IS NO OPERATOR IN FORMULA ALGOL WHICH DENOTES DIRECTLY A GAMMA OR A BETA FUNCTION, BUT BY MEANS OF THE '.(' CONSTRUCTION A REPRESENTATION CAN BE OBTAINED. FIRST, WE DECLARE TWO PROCEDURES:

```
FORM PROCEDURE GAMMA(N); VALUE N; FORM N;
GAMMA ← GAMMA.(N);
FORM PROCEDURE BETA(N,M); VALUE M,N; FORM M,N;
BETA ← BETA.(N,M);
```

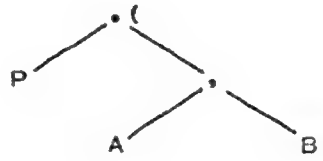
AFTER HAVING DECLARED SUCH PROCEDURES WE CAN USE THEM IN ANY FORMULA; FOR INSTANCE

(X + BETA.(A,B))/GAMMA.(Q↑2)

IS A FORMULA WITH THE TREE STRUCTURE:



IN GENERAL, IF P IS A DECLARED FUNCTION DESIGNATOR (PROCEDURE WITH VALUE) THEN THE CONSTRUCTION P(A,B) IS A CALL TO THE PROCEDURE P WITH ACTUAL PARAMETERS A AND B, THE VALUE OF SUCH A CONSTRUCTION WOULD BE WHATEVER IS PRODUCED FROM THE PROCEDURE BODY OF P. ON THE OTHER HAND IF THE CONSTRUCTION IS P.(A,B) NO CALL TO P IS MADE, THE VALUE WOULD BE ''A REPRESENTATION OF THE CALL'' OR IN PROGRAMMING TERMS A POSTPONED CALL REPRESENTED BY THE TREE



THE LANGUAGE HAS A LARGE COLLECTION OF FEATURES WHICH WILL PERMIT AN ADEQUATE MANIPULATION OF THE REPRESENTATION OF FUNCTION DESIGNATORS. FOR INSTANCE, ONE COULD WRITE A SIMPLIFICATION ROUTINE BASED ON PRODUCTIONS WHICH MIGHT LOOK LIKE

```
GAMMA.(N:OF(GREATHAN1))   →   (.N-1)*GAMMA.(.N-1)
BETA.(N:FORM, M:FORM)     →   GAMMA.(.N)*GAMMA.(.M)/GAMMA.(N+M)
```

ETC.; OR IF EVENTUALLY A NUMERIC VALUE WERE DESIRED THE PROCEDURE COULD

BE DECLARED AS FOLLOWS

```
FORM PROCEDURE GAMMA(M); VALUE M; FORM M;
IF M==FORM THEN GAMMA ← GAMMA.(M) ELSE
IF M¬>0 THEN PRINT( 'GAMMA(M) UNDEFINED WHEN N¬>0' ) ELSE
BEGIN
COMMENT: ASSIGN TO GAMMA THE VALUE OBTAINED BY COMPUTING
THE NUMERIC APROXIMATION TO THE INTEGRAL:
```

$$\int_0^{\infty} x^{M-1} e^{-x} dx \;;$$

```
...

END;
```

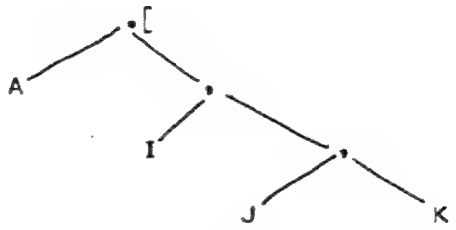
THUS EVERY TIME A FORMULA CONTAINING A GAMMA OR A BETA FUNCTION IS EVALUATED, IT WILL PRODUCE A NUMBER ONLY IF THE ACTUAL PARAMETER IS A NUMBER, OTHERWISE IT WILL REMAIN AS A REPRESENTATION OF THE FUNCTION.

ONCE THE LANGUAGE IS EXPANDED TO INCLUDE REPRESENTATIONS OF FUNCTION DESIGNATORS AND REPRESENTATIONS OF CONDITIONAL EXPRESSIONS BY DEFINING A DOT PREFIX FOR '(' AND 'IF', IT IS CLEAR THAT, FROM THE PROGRAMMING VIEWPOINT, THE DOT MEANS A POSTPONED ACTION, A WAY FOR TALKING ABOUT THE NAME OF AN ACTION RATHER THAN THE ACTION ITSELF. THEREFORE IT WAS A NATURAL STEP TO GENERALIZE THIS CONCEPT AND DEFINE THE DOT PREFIX FOR OTHER CONSTRUCTIONS.

FIRSTLY, THE POSTPONED ARRAY ACCESS WAS PROVIDED, FOR EXAMPLE

A.[I,J,K]

IS REPRESENTED BY THE TREE



SECONDLY, THE POSTPONED ASSIGNMENT WAS PROVIDED, FOR EXAMPLE,

G •← F

WHICH IS REPRESENTED BY THE TREE

FINALLY, THE POSTPONED SIMPLE VARIABLE ACCESS WAS PROVIDED, FOR EXAMPLE,

•X

MAY BE USED. WHEN THE SIMPLE VARIABLE X IS USED IN AN EXPRESSION, IT MEANS THE VALUE OF X, I.E., THERE IS AN ACCESS TO THE CURRENT VALUE OF X. ON THE OTHER HAND, IF ONE WANTS TO TALK ABOUT X ITSELF AND NOT ITS CURRENT VALUE, THEN WE POSTPONE THE ACCESS BY WRITTING '•X'. WHEN A VARIABLE IS DECLARED OF TYPE FORM ITS VALUE IS INITIALIZED TO ITS OWN NAME.

THE AUTHOR HAS NOT FOUND ANY INTERESTING APPLICATION FOR POSTPONED ASSIGNMENTS. ON THE OTHER HAND THE POSTPONED ACCESS, BOTH FOR ARRAYS AND SIMPLE VARIABLES, TURNED OUT TO BE A VERY USEFUL FEATURE WHICH, AMONG OTHER THINGS, PROVIDES AN ANSWER FOR THE FOLLOWING PROBLEM: WHILE DOING SOME ALGEBRAIC MANIPULATIONS ONE MAY BE FORCED TO BUILD THE SKELETON OF A FORMULA, I.E., AT A GIVEN MOMENT ONE KNOWS ONLY THE MAIN FUNCTIONAL RELATIONS (CONNECTORS) OF A FORMULA, BUT YET ONE IGNORES THE DETAILS OF SOME SUBFORMULAE (BRANCHES). ONE CAN GIVE NAMES TO UNKNOWN

PARTS (I.E. POSTPONE THE ACCESS) AND USE THOSE NAMES IN CONSTRUCTING THE FORMULA. LATER, WHEN THE DETAILS IN THE BRANCHES ARE FILLED OUT, ONE CARRIES OUT THE ACCESS TO THE ACTUAL VALUES. THIS TECHNIQUE IS SHOWN IN AN ITERATIVE ALGORITHM FOR DIFFERENTIATION (APPENDIX-2).

THESE ARE ALL THE FORMULA DATA STRUCTURES EXISTING AS OF NOW IN THE FORMULA ALGOL LANGUAGE. FACULTY AND STUDENTS IN CARNEGIE TECH'S COMPUTER SCIENCE DEPARTAMENT HAVE MADE A NUMBER OF GOOD SUGGESTIONS FOR NEW FORMULA DATA STRUCTURES AND PRIMITIVE OPERATIONS. THESE SUGGESTIONS ARE NOW BEING STUDIED FOR THEIR POSSIBLE INCLUSION IN A NEW VERSION OF FORMULA ALGOL. THESE NEW DEVELOPMENTS WILL BE REPORTED AT A LATER TIME.

FORMULA ALGOL HAD ORIGINALLY ONLY FORMULA DATA STRUCTURES BESIDES THOSE OF ALGOL-60. AS A RESULT OF OUR FIRST RUNNING SYSTEM (SUMMER 1963) WE REALIZED THAT WE HAD PRODUCED INDEED A TREMENDOUS GAIN IN POWER TO DO ALGEBRAIC MANIPULATIONS, BUT SUCH POWER WAS LIMITED BY A LACK OF GENERAL TREE STRUCTURES TO PROVIDE THE NECESSARY ORGANIZATIONAL FRAMEWORK FOR SOPHISTICATED ALGORITHMS. E.G., THE ADMINISTRATION OF A HEURISTIC SEARCH, THE GOAL TREE TECHNIQUE, THE DISCRIMINATION NETS, THE LEARNING DEVICES, ETC. ALL REQUIRE FOR THEIR NATURAL IMPLEMENTATATION THE EXISTENCE OF ARBITRARY LIST STRUCTURES. THESE REASONS, BESIDES THOSE STATED EARLIER IN THIS CHAPTER, LED TO THE ADDITION OF GENERAL SYMBOLIC PROCESSING CAPABILITIES TO THE LANGUAGE.

ONCE THERE WERE GENERAL TREE STRUCTURES IN THE LANGUAGE, WE USED THEM TO IMPROVE SOME OF THE FORMULA MANIPULATION CAPABILITIES AS WE WILL SEE AT VARIOUS POINTS IN THE REMAINING PART OF THIS CHAPTER. THE FIRST

DIRECT INFLUENCE ON THE FORMULA DATA STRUCTURES, PRODUCED BY THE GENERAL LIST ADDITION, WAS THE INCLUSION OF DESCRIPTION LISTS FOR FORMULAE. TO EACH DECLARED FORMULA VARIABLE THERE IS ATTACHED A 'DESCRIPTION LIST' WHICH IS A LIST OF PAIRS, THE FIRST MEMBER OF EACH PAIR IS CALLED THE ATTRIBUTE, THE SECOND IS CALLED THE VALUE. BY MEANS OF LIST PRIMITIVES, WE CAN TEST, ADD, DELETE, OR ALTER ATTRIBUTE VALUE PAIRS. THIS GAVE US THE ABILITY TO DESCRIBE FORMULAE, FOR INSTANCE WE COULD HAVE THE ATTRIBUTES 'DIFFERENTIABLE', 'CONTINUOUS', 'P-SATISFYING', ETC.

SOME FACTS ABOUT FORMULA DATA STRUCTURES IMPLEMENTATION. FORMULAE ARE REPRESENTED INSIDE THE COMPUTER MEMORY BY MEANS OF LIST STRUCTURES, ALTHOUGH, TO ACHIEVE STORAGE EFFICIENCY FORMULAE ARE REPRESENTED DIFFERENTLY THAN GENERAL SYMBOLIC STRUCTURES. IN ANY CASE, FOR BOTH THERE IS AN AVAILABLE SPACE LIST WHICH IS THE GENERAL SOURCE OF STORAGE. THIS AVAILABLE SPACE LIST IS NOT PART OF THE ALGOL BLOCK STORAGE ALLOCATION. FOR THE LATTER, THERE IS A RUN-TIME SYMBOL TABLE WHICH CONTAINS ALL THE VARIABLES CURRENTLY DEFINED. THIS TABLE MAY GROW OR DIMINISH AS THE FLOW OF CONTROL ARRIVES AT BLOCK 'BEGIN'S OR 'END'S, RESPECTIVELY.

THE VALUE OF A FORM VARIABLE IS A POINTER TO A LIST STRUCTURE. WHEN A FORMULA IS CONSTRUCTED AND ASSIGNED TO A VARIABLE NO COPIES OF BRANCHES ARE MADE, FOR EXAMPLE IN THE ASSIGNMENT

F ← A + B*C;

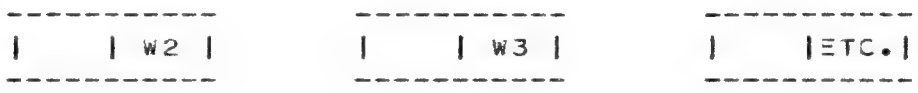
NO COPIES OF A, B OR C ARE MADE AND ONLY TWO CELLS FROM AVAILABLE SPACE ARE USED. SUPPOSE THAT BEFORE THE ASSIGNMENT THE SYMBOL TABLE LOOKED LIKE:

F: ALPHA
A: BETA
B: GAMMA
C: DELTA

AND THE HEAD OF AVAILABLE SPACE L WAS:

L: W1

WHERE



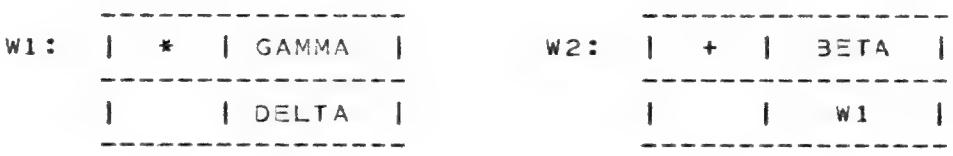
AFTER THE ASSIGNMENT, THE SYMBOL TABLE LOOKS LIKE:

F: W2
A: BETA
B: GAMMA
C: DELTA

THE HEAD OF AVAILABLE SPACE IS NOW:

L: W3

THE TWO CELLS TAKEN FROM AVAILABLE SPACE ARE:



THE OLD VALUE OF F, I.E., THE POINTER ALPHA, MAY STILL POINT TO A STRUCTURE ATTACHED TO SOME FORMULA CURRENTLY DEFINED OR IT MAY BE POINTING TO A PIECE OF STRUCTURE THAT WAS LOST BY THE ASSIGNMENT. WHEN THERE IS A SHORTAGE OF AVAILABLE SPACE A 'GARBAGE COLLECTOR' ROUTINE WI__ RECOVER THE LOST PIECES RETURNING THEM TO THE AVAILABLE SPACE.

EVALUATION AND SUBSTITUTION.

FORMULAE ARE THE REPRESENTATION OF MATHEMATICAL FUNCTIONS. THEY DESCRIBE A MAPPING OF N-TUPLES OF VALUES INTO A SINGLE VALUE. IN ONE SENSE WE MAY THINK OF FORMULAE AS ABSTRACTIONS OF COMPUTATIONS, IN THE CASE THAT WE INTEND TO SUBSTITUTE NUMBERS FOR VARIABLES; IN ANOTHER SENSE, WE MAY THINK OF FORMULAE AS FUNCTIONAL RELATIONS, IN THE CASE THAT WE INTEND TO SUBSTITUTE FORMULAE FOR VARIABLES. IN ANY CASE BY MANIPULATING FORMULAE, WE ALTER THE MAPPING THEY REPRESENT. AT SOME POINT IN THE EXECUTION OF A PROGRAM, WE MAY WISH TO CARRY OUT THE MAPPING REPRESENTED BY A FORMULA. TO DO THIS, WE COULD SUBSTITUTE VALUES FOR OCCURRENCES OF VARIABLES APPEARING BY NAME ONLY IN A FORMULA, AND THESE VALUES WOULD BE COMBINED ACCORDING TO THE MAPPING RULES EXPRESSED BY THE FORMULA AND WOULD RESULT IN AN EVALUATED FORMULA. IN ORDER TO ACCOMPLISH THE ABOVE, WE HAVE THE 'EVAL', THE 'SUBS' AND THE 'REPLACE' OPERATORS.

THE EVAL EXPRESSION IS BASED ON LAMBDA NOTATION [9], FOR EXAMPLE CONSIDER THE EXPRESSION:

EVAL(A,B)F(3,4)

WHICH MEANS EVALUATE THE FORMULA F ASSIGNING THE VALUE 3 TO ALL OCCURRENCES OF THE VARIABLE A IN F AND ASSIGNING THE VALUE 4 TO ALL OCCURRENCES OF THE VARIABLE B IN F.

A GOOD EXAMPLE SHOWING THE USE OF NUMERICAL COMPUTATION COMBINED WITH ALGEBRAIC SYMBOLIC MANIPULATIONS IS THE EXPANSION OF A FUNCTION BY

TAYLOR SERIES. IN THIS EXAMPLE, THE INPUT IS A FORMULA (A FUNCTION TO BE EXPANDED), THE ALGORITHM HAS TO ACCOMPLISH SYMBOLIC COMPUTATIONS (OBTAIN DERIVATIVES), AND HAS TO CARRY OUT NUMERICAL COMPUTATIONS BY MAPPING THE FORMULAE PRODUCED INTO NUMBERS (EVALUATION OF DERIVATIVES), AND FINALLY IT HAS TO COMBINE ALL THESE RESULTS IN AN ALGEBRAIC FORMULA (A POLYNOMIAL) WHICH IS THE RESULT. WE HAVE,

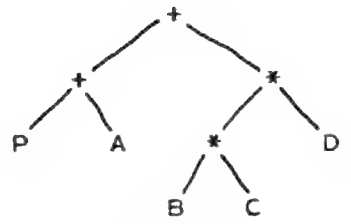
$$\sum_{i=0}^{N} \frac{g^{[i]}(x)\big|_{x=A}}{i!} (X-A)^{i}$$

IN FORMULA ALGOL

```
T ← 0;
G ← F;
FOR I←0 STEP 1 UNTIL N DO BEGIN
T ← T + EVAL(X)G(A)/FACTORIAL(I)*(X-A)↑I;
G ← DERV(G,X) END;
```

WHERE T, X, G AND F ARE FORM VARIABLES, A IS A REAL VARIABLE, N AND I ARE INTEGER VARIABLES, FACTORIAL IS AN INTEGER PROCEDURE AND DERV IS A FORM PROCEDURE. AFTER THE 'FOR LOOP' HAS BEEN EXECUTED T WILL CONTAIN THE FIRST N+1 TERMS OF THE TAYLOR SERIES EXPANSION OF F IN THE NEIGHBORHOOD OF X=A.

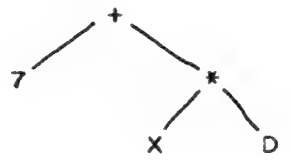
DURING AN EVALUATION IT MIGHT BE THAT NOT ALL VARIABLES WERE REPLACED BY NUMBERS OR SOME VARIABLES WERE REPLACED BY FORMULAE. IN THESE CASES THE RESULT WILL BE A FORMULA. SOME OF THE BRANCHES MIGHT BE COLLAPSED TO A NUMBER, AND IN SUCH CASES SOME AUTOMATIC REDUCTIONS CAN TAKE PLACE. FOR EXAMPLE, LET F LOOK LIKE



I.E., F COULD HAVE BEEN PRODUCED BY THE STATEMENT:

F ← (P+A) + (B*C)*D;

WHEN THE STATEMENT G ← EVAL(P,A,B,C)F(3,4,1,X); IS EXECUTED, G WOULD HAVE THE FOLLOWING TREE STRUCTURE



THE AUTOMATIC SIMPLIFICATIONS CARRIED OUT BY EVAL ARE:

A ↑ 0 → 1	A * 0 → 0	
A ↑ 1 → A	A * 1 → A	
A ↑-1 → 1/A	A *-1 → -A	COMMUTATIVE
A ↑-N → 1/A↑N	A *-N → -(A*N)	
A / 1 → A	A + 0 → A	
A /(-1)→ -A	A + (-N) → A - N	
A /(-N)→ -(A/N)	A - (-N) → A + N	
0 / A → 0	A - 0 → A	
(-N) / A → -(N/A)	0 - A → -A	

X ∨ TRUE → TRUE
X ∧ TRUE → X
X ∨ FALSE → X
X ∧ FALSE → FALSE

A LAST PROPERTY OF EVAL IS THE ACCOMPLISHMENT OF POSTPONED ACTIONS DELAYED BY A DOT-CONSTRUCTION.

THERE ARE SOME WEAK POINTS IN THE 'EVAL' DESCRIBED SO FAR. IN

THE FIRST PLACE, THERE IS NO REASON WHY ONE MUST SPECIFY EXPLICITLY THE BOUND VARIABLES. ONE MAY NOT KNOW WHICH VARIABLES WOULD EVEN BE IN THE FORMULA. ONCE WE HAD A GENERAL LIST STRUCTURE AS PART OF THE LANGUAGE, THE SOLUTION WAS CLEAR: THE NAME OF A LIST COULD BE USED INSTEAD OF THE LIST OF SPECIFIC BOUND VARIABLES. THUS WE EXTEND THE SYNTAX TO INCLUDE CASES LIKE:

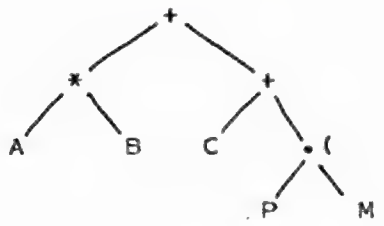
EVAL([A])F([B])

WHERE A AND B ARE THE NAMES OF LISTS. BEFORE THE ABOVE EXPRESSION IS EXECUTED, THE CONTENTS OF A AND B CAN BE DYNAMICALLY CREATED OR CHANGED. THE SQUARE BRACKETS AROUND A AND B INDICATE THEY ARE NAMES OF LISTS, OTHERWISE THE EXPRESSION WOULD BE AMBIGUOUS --SINCE EVAL(A)F(B) MEANS EVALUATE F REPLACING ALL OCCURRENCES OF THE ATOMIC FORMULA A BY THE VALUE OF FORMULA VARIABLE B.

A SECOND OBJECTION WAS THAT SINCE FORMULAE CAN HAVE DIFFERENT INTERPRETATIONS FROM THOSE GIVEN IN STANDARD ANALYSIS, THE AUTOMATIC SIMPLIFICATIONS AND EVALUATIONS COULD BE HARMFUL AND MEANINGLESS. FOR THIS REASON WE CREATED THE 'PURE SUBSTITUTION' CONSTRUCTIONS, FOR EXAMPLE,

SUBS(A,B)F(3,X)
SUBS([P])F([Q])

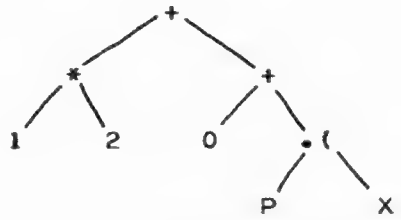
THESE EXPRESSIONS ALWAYS CONSTRUCT A FORMULA, NO NUMERIC EVALUATION IS EXECUTED, NO SIMPLIFICATIONS ARE MADE, AND NO POSTPONED ACTIONS ARE CARRIED OUT. SUPPOSE F LOOKS LIKE:



WHEN THE STATEMENT

```
G ← SUBS(A,B,C,M)F(1,2,0,X);
```

IS EXECUTED, G CHANGES TO:



IT SHOULD BE UP TO THE USER TO DEFINE EVALUATION RULES FOR ANY SPECIAL INTERPRETATION HE CHOOSES FOR HIS FORMULAE. THE LANGUAGE HAS PRIMITIVES THAT WILL ALLOW HIM TO PROGRAM ANY EVALUATION PROCEDURE. THE 'SUBS' MECHANISM GIVES HIM A QUICK WAY TO ACCOMPLISH A STRAIGHT FORWARD SUBSTITUTION WITHOUT ANY 'A PRIORI' INTERPRETATION OF HIS FORMULAE.

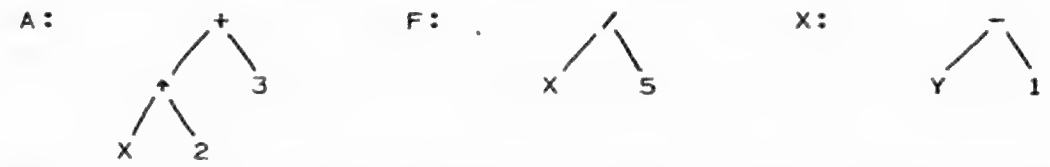
A LAST ISSUE RELATED TO EVAL AND SUBS IS THAT OF A NON RETROACTIVE ASSIGNMENT -- AN IMPORTANT CONCEPT IN ALGOL-60 AND IN ALL NUMERICALLY ORINTED LANGUAGES. FOR EXAMPLE, CONSIDER THE FOLLOWING PIECE OF PROGRAMMING.

```
F ← X↑2;
A ← F+3:
F ← X/5;
X ← Y-1;
```

THE VALUE OF A WAS CREATED USING THE VALUE OF F WHICH ITSELF WAS CREATED USING THE VALUE OF X. THE FACT THAT F AND X WERE ALTERED LATER DOES NOT IMPLY A CHANGE IN THE VALUE OF A. WITH NUMERIC COMPUTATIONS THIS IS THE

CASE BY DEFAULT, SINCE IT WOULD BE IMPOSSIBLE TO RECOGNIZE JUST BY LOOKING AT THE NUMBER 4.5 THAT IT WAS CREATED AS THE RESULT OF THE ADDITION OF 3.75 AND 0.75. ALTHOUGH WITH ALGEBRAIC MANIPULATIONS IT MAY HAVE MEANING AND SOME TIMES IT WOULD BE POSSIBLE TO ACCOMPLISH RETROACTIVE ASSIGNMENTS. THE NORMAL 'MODUS OPERANDI' IN FORMULA ALGOL IS NON RETROACTIVE.

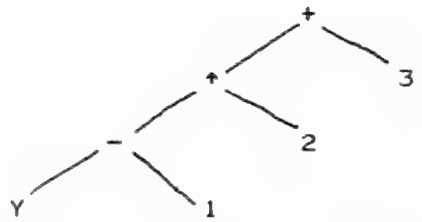
NEVERTHELESS BY MEANS OF EVAL OR SUBS, RETROACTIVE ASSIGNMENTS CAN SOMETIMES BE ACHIEVED. FOR EXAMPLE, CONSIDER THE ABOVE PIECE OF PROGRAMMING. LET F, A, X AND Y BE FORM VARIABLES; LET THE VALUE OF X AND Y BE THEMSELVES BEFORE THE FIRST STATEMENT IS EXECUTED. AFTER THE LAST STATEMENT IS EXECUTED THE VARIABLES WOULD BE:



WE SEE THAT THE FACT THAT F WAS USED IN THE CONSTRUCTION OF A HAS BEEN LOST. BUT THE VALUE OF A STILL RECALLS THE FACT THAT X WAS USED (IMPLICITLY) IN ITS CONSTRUCTION. THE STATEMENT

```
A ← SUBS(.X)A(X);
```

WILL ACHIVE A RETROACTIVE ASSIGNMENT, I.E. IT WILL UPDATE THE VALUE OF A, WHICH NOW LOOKS LIKE:



IN GENERAL, ONE DOES NOT KNOW THE NAMES OF THE VARIABLES OCCURRING IN A

FORMULA. SO THAT THE ABOVE METHOD CANNOT BE USED TO ACHIEVE RETROACTIVE ASSIGNMENTS. FOR THIS REASON WE CREATED THE REPLACE OPERATOR. THE EXPRESSION:

REPLACE(F)

WHERE F IS A FORMULA, PRODUCES ANOTHER FORMULA WHICH IS OBTAINED FROM F BY REPLACING EVERY ATOMIC VARIABLE IN F BY ITS CURRENTLY ASSIGNED VALUE. THE ATOMIC VARIABLES USED IN THE FORMULA F MUST BE DECLARED EITHER LOCALLY OR GLOBALLY TO THE BLOCK IN WHICH REPLACE IS EXECUTED.

FORMULA PREDICATES AND EXTRACTORS.

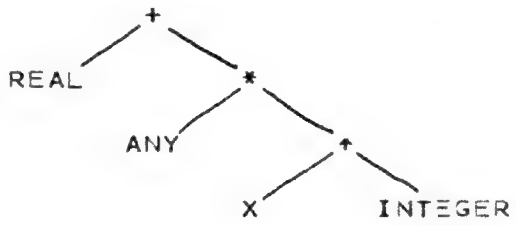
A COMPUTATION MAY CONSTRUCT A FORMULA WHOSE STRUCTURE CANNOT BE PREDICTED IN ADVANCE OR A SITUATION MAY ARISE IN A PROGRAM WHERE IT IS DESIRED TO DISCRIMINATE AMONG VARIOUS FORMULAE IN A GIVEN CLASS DEPENDING ON THEIR VARIOUS PROPERTIES. FOR THIS A MECHANISM IS NEEDED TO DETERMINE PRECISELY THE STRUCTURE OF ANY GIVEN FORMULA. FORMULA PATTERNS ARE USED FOR THIS PURPOSE AND THEY CONSTITUTE A SET OF PREDICATES OVER THE CLASS OF FORMULA DATA STRUCTURES. THESE FORMULA PATTERNS ARE SUFFICIENT IN THE SENSE THAT WHATEVER CONSTRUCTIONS ARE USED TO CREATE A FORMULA MAY BE REVERSED BY A CHOICE OF A SEQUENCE OF PREDICATES. FURTHEMORE, A GIVEN FORMULA PATTERN MAY BE USED TO REPRESENT A CLASS OF POSSIBLE FORMULAE, AND ANY FORMULA MAY BE TESTED FOR MEMBERSHIP IN THIS CLASS.

PATTERNS.

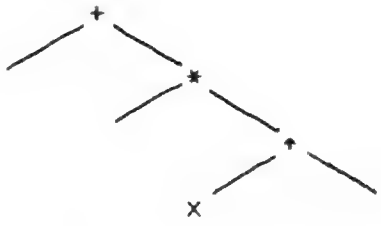
PATTERNS ARE DATA STRUCTURES THAT DEFINE CLASSES OF FORMULAE, FOR EXAMPLE

REAL + ANY * X↑INTEGER

IS A PATTER THAT DEFINES THE CLASS OF ALL FORMULAE WHOSE MAIN CONNECTOR IS THE OPERATOR '+', WHOSE LEFT BRANCH IS A REAL NUMBER AND WHOSE RIGHT BRANCH IS A FORMULA WHICH IS THE PRODUCT OF ANY FORMULA MULTIPLIED BY X RAISED TO ANY INTEGER POWER. PATTERNS HAVE A TREE STRUCTURE, THE SAME AS FORMULAE. THUS, THE ABOVE EXAMPLE HAS THE TREE STRUCTURE



A FORMULA IS A PATTERN WHICH DEFINES A CLASS OF ONE MEMBER, NAMELY ITSELF. OBSERVING THE ABOVE TREE STRUCTURE WE NOTICE THAT THE PATTERN WAS FORMED FROM A SYNTACTIC SKELETON, GIVEN BY THE STRUCTURE



WHICH HAS SOME OF ITS END NODES (LEAVES) DESCRIBING A SEMANTIC PROPERTY:

REAL, ANY, INTEGER.

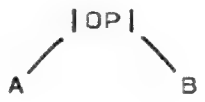
IN GENERAL, PATTERNS ARE AN AMALGAMATE OF A SYNTACTIC STRUCTURE WITH SEMANTIC DESCRIPTIONS.

PATTERNS HAD A CONSIDERABLE EVOLUTION DURING THE FOUR YEARS OF FORMULA ALGOL'S DEVELOPMENT. WE MADE CHANGES THAT ALLOWED 1) MORE FLEXIBLE SYNTACTIC STRUCTURES AND 2) MORE POWERFUL SEMANTIC DESCRIPTIONS.

CONSIDER THE FIRST GOAL: INCREASING THE FLEXIBILITY IN THE PATTERN'S SYNTACTIC STRUCTURES. THE NODES OF A PATTERN STRUCTURE ARE OPERATORS. BY USING ARBITRARY NAMES (SYMBOL VARIABLES) INSTEAD OF OPERATORS IN A PATTERN STRUCTURE WE GAINED CONDENSED NOTATION TO DESCRIBE GREATER VARIABILITY. WE CALLED THOSE NAMES ''CLASS OPERATORS''S, FOR FOR EXAMPLE

A |OP| B

WHICH DENOTES A PATTERN WHOSE MAIN CONNECTOR IS THE CLASS OPERATOR 'OP', WHOSE LEFT MEMBER IS THE FORMULA A AND WHOSE RIGHT MEMBER IS THE FORMULA B. IT HAS THE TREE STRUCTURE:



NOW, ONE CAN DYNAMICALLY ASSIGN MANY MEANINGS TO THE OPERATOR 'OP'. ONE CAN ACCOMPLISH THIS THROUGH THE USE OF SPECIAL ATTRIBUTES CREATED FOR THE CLASS OPERATORS. FOR INSTANCE ONE COULD SAY

OP ← /[OPERATOR:+];

IN WHICH CASE THE ABOVE PATTERN WOULD DEFINE THE FORMULA A+B BUT EVEN MORE, ONE CAN ASSIGN A COLLECTION OF OPERATORS

OP ← /[OPERATOR:+,-,*,/,];

IN THIS CASE ANY OF THE FORMULAS: A+B, A-B, A*B OR A/B WOULD BE A MEMBER OF THE PATTERN A |OP| B.

COMMUTATIVITY PLAYS AN IMPORTANT ROLE IN SOME ALGEBRAS. FOR THIS

REASON WE DECIDED TO INCREASE THE FLEXIBILITY OF PATTERNS, BY INTRODUCING A COMMUTATIVE ATTRIBUTE WITH THE CLASS OPERATORS. LET US CONSIDER THE ASSIGNMENT:

```
OP ← /[OPERATOR:+,-,*,/][COMM:TRUE,FALSE,TRUE,FALSE];
```

NOW ANY ONE OF THE FORMULAS A+B, B+A, A-B, A*B, B*A OR A/B WOULD BE A MEMBER OF A |OP| B.

THERE IS ANOTHER SPECIAL ATTRIBUTE FOR CLASS OPERATORS: INDEX. IT WAS CREATED AS AN ANSWER TO THE FOLLOWING EFFICIENCY PROBLEM: MANY ALGEBRAIC MANIPULATION ALGORITHMS START BY SELECTING A SUB-ALGORITHM DEPENDING ON THE MAIN CONNECTOR OF THE INPUT FORMULA, USING A CHAIN OF IF-THEN-ELSE'S IN WHICH IS ASKED: IS THE INPUT FORMULA'S MAIN CONNECTOR THE OPERATOR '+'. IF YES THEN ... ELSE IS THE INPUT FORMULA'S MAIN CONNECTOR THE OPERATOR '-'. IF YES THEN ... ELSE ETC. HERE ONE NEEDS AS MANY PATTERNS AS OPERATORS TO BE TESTED. ON THE OTHER HAND, ONE CAN DEFINE A CLASS OPERATOR CONTAINING ALL POSSIBLE MAIN CONNECTORS, SAY

```
OP ← /[OPERATOR:+,-,*,/,↑][INDEX:I];
```

THE PATTERN

```
ANY |OP| ANY
```

CHARACTERIZES ANY FORMULA WHOSE MAIN CONNECTOR IS EITHER '+', '-', '*', '/' OR '↑'. FURTHERMORE WHEN ONE ASKS WHETHER SOME FORMULA IS A MEMBER OF THAT PATTERN THE SPECIAL ATTRIBUTE 'INDEX' (WHOSE VALUE MUST BE A DECLARED INTEGER VARIABLE) INDICATES THAT THE VARIABLE 'I' WILL BE ASSIGNED AN INTEGER 1, 2, 3, 4 OR 5, DEPENDING ON WHETHER THE MAIN CONNECTOR WAS '+', '-', '*', '/', OR '↑', RESPECTIVELY. THE VALUE OF I CAN THEN BE USED, FOR EXAMPLE, TO INDEX A SWITCH DESIGNATOR THAT SELECTS

THE APPROPRIATE ALGORITHM.

WITH THIS WE CONCLUDE THE EXPLANATION OF FLEXIBILITY IN PATTERN'S SYNTACTIC STRUCTURES. NOW LET US EXAMINE THE IMPROVEMENTS IN THE SCOPE OF SEMANTIC DESCRIPTIONS WITHIN PATTERNS.

ORIGINALLY FOR SEMANTIC DESCRIPTIONS WE HAD ONLY THE 'TYPE' WORDS:

REAL, INTEGER, BOOLEAN, FORM.

WE EXTEND THEM TO INCLUDE THE WORDS:

ATOM, ANY, LOGICAL.

THUS FAR THE DESCRIPTIONS, FIXED AT COMPILE TIME, HAVE LIMITED POWER. ONE MUST BE ABLE TO CHANGE DESCRIPTIONS OR ASSIGN THEM DYNAMICALLY DURING THE ACTUAL RUN OF A PROGRAM. TO ACCOMPLISH THIS, THE SOLUTION WAS ALREADY CLEAR: GIVE A NAME TO SUCH SEMANTIC DESCRIPTION. THE CONSTRUCTION

OF(Q) + REAL

IS A PATTERN DEFINING FORMULAE WHICH AMONG OTHER THINGS ARE REQUIRED TO HAVE AS LEFT BRANCH A MEMBER OF THE CLASS DEFINED BY Q, WHERE Q IS A SYMBOL VARIABLE. THUS ONE ASSIGN A LIST OF PATTERNS TO Q, FOR EXAMPLE,

Q + [REAL+FORM, INTEGER/X, FORM |T| REAL];

THE PATTERN 'OF(Q)' DEFINES THE CLASS OF FORMULAE WHICH ARE THE UNION OF ALL FORMULAE WHICH ARE MEMBERS OF ANY OF THE CLASSES DEFINED BY ELEMENTS OF Q.

A FINAL EXTENSION THAT DELIVERED ALL THE POWER OF FORMULA ALGOL ITSELF INSIDE THE PATTERN STRUCTURES IS THE FOLLOWING EXTENSION IN THE SEMANTICS OF THE CONSTRUCTION

OF(P)

WHERE 'P' IS THE NAME OF A BOOLEAN PROCEDURE. THE MEANING OF THIS CONSTRUCTION IS THE PATTERN THAT DEFINES THE CLASS OF FORMULAE FOR WHICH THE PROCEDURE 'P' IS TRUE.

LET US CONSIDER THE FOLLOWING EXAMPLE. ONE IS WORKING WITH LOGICAL FORMULAE CONSTRUCTED WITH TWO VARIABLES P AND Q. SUPPOSE THAT FOR SOME REASON ONE WANTS TO USE THE CLASS OF FORMULAE WHOSE MEMBERS ARE A CONJUCTION OF P WITH ANY TAUTOLOGY. THIS IS DESCRIBED IN FORMULA ALGOL WITH THE PATTERN

P ^ OF (TAUTOLOGY)

WHERE THE BOOLEAN PROCEDURE 'TAUTOLOGY' IS DEFINED:

```
BOOLEAN PROCEDURE TAUTOLOGY(F); VALUE F; FORM F;
BEGIN FORM A,B;
FOR A←TRUE, FALSE DO
FOR B←TRUE, FALSE DO
IF ¬EVAL(P,Q)F(A,B) THEN GOTO NO;
TAUTOLOGY ← TRUE; GOTO FIN;
NO: TAUTOLOGY ← FALSE;
FIN: END;
```

FORMULA PREDICATES.

PATTERN STRUCTURES ARE USED IN SEVERAL WAYS IN THE LANGUAGE. FIRST OF ALL, THEY ARE USED IN PREDICATES FOR FORMULAE. THERE ARE TWO FORMULA PREDICATES: THE 'INSTANCE' PREDICATE DENOTED BY '==' AND THE 'CONTAINS' PREDICATE DENOTED BY '>>'. THE EXPRESSION

F == P

IS A BOOLEAN PRIMARY, I.E., IT HAS A BOOLEAN VALUE AND IT CAN BE USED AS

AN ELEMENT IN ANY BOOLEAN EXPRESSION. IT IS READ AS FOLLOWS: 'IS THE FORMULA F AN INSTANCE OF THE PATTERN P'. FOR EXAMPLE IF

```
P ← FORM↑INTEGER + REAL;
F ← X   ↑   2    + 3.5 ;
```

THEN THE BOOLEAN EXPRESSION 'F==P' IS TRUE. THE EXPRESSION

```
F >> Q
```

IS ALSO A BOOLEAN PRIMARY AND IS READ AS FOLLOWS: 'DOES THE FORMULA F CONTAIN A BRANCH WHICH IS AN INSTANCE OF Q'. FOR EXAMPLE, WHEN

```
Q ← FORM↑2;
```

THE BOOLEAN EXPRESSION 'F>>Q' IS TRUE. SINCE F=X↑2+3.5 HAS A BRANCH, NAMELY X↑2, WHICH IS AN INSTANCE OF FORM↑2.

EXTRACTORS

THE USEFULNESS OF FORMULA PREDICATES WOULD BE VERY LIMITED IF WE COULD NOT ISOLATE THE DIFFERENT BRANCHES OF A FORMULA ONCE THEY ARE RECOGNIZED. FOR THIS PURPOSE WE HAVE EXTRACTORS THAT ASSIGN TO VARIABLES THE ACTUAL VALUES OF DIFFERENT PARTS OF A FORMULA. THIS IS DONE BY LABELING THE PATTERN STRUCTURE. THUS, FOR INSTANCE THE ASSIGNMENT

```
P1 ← A:FORM ↑ I:INTEGER + C:REAL
```

CREATES A PATTERN P1 WHICH DEFINES EXACTLY THE SAME CLASS OF FORMULAE DEFINED BY P ABOVE, BUT P1 HAS LABELED ITS BRANCHES. WHEN THE PATTERN P1 IS USED IN A PREDICATE, SAY

```
F == P1
```

THE PREDICATE IS NOT ONLY TRUE, BUT PRODUCES A SIDE EFFECT: THE VARIABLES

A, I AND C ARE ASSIGNED THE VALUES X, 2 AND 3.5, RESPECTIVELY.

THERE ARE SEVERAL BASIC ACTIVITIES IN ALGEBRAIC MANIPULATION THAT CAN BE DESCRIBED IN FORMULA ALGOL USING THE PREDICATE '>>'.

THE FIRST SUCH ACTIVITY COULD BE DESCRIBED AS: ''TO FOCUS ONE'S ATTENTION'' AT SOME PART OF A GIVEN FORMULA. BEFORE A FOCUSSING ACTION THERE IS A SEARCH ACTION, I.E., ONE SEEKS A FORMULA WHICH HAS CERTAIN PROPERTIES TO CONCENTRATE THE COMPUTATIONAL ACTIVITY ON IT. THERE IS IN THE LANGUAGE A PRIMITIVE OPERATION TO PERFORM SUCH A PROCESS. THE CONSTRUCTION

F >> V:P

IS A PREDICATE WHICH IS TRUE WHEN F>>P IS. FURTHERMORE IF THIS IS THE CASE, THE FIRST SUB-EXPRESSION (SEEKING FROM RIGHT TO LEFT) OF THE FORMULA F THAT MATCHES THE PATTERN P WILL BE ASSIGNED TO V, FROM WHERE IT CAN BE BE PROCESSED FURTHER.

ANOTHER ACTIVITY IN FORMULA MANIPULATION IS THAT OF ''CONSIDERING ONLY THE ESSENTIAL CHARACTERISTICS'' OF A FORMULA. HERE WE DESIRE TO PRUNE A TREE STRUCTURE, TO CUT AWAY IRRELEVANT SUB-EXPRESSIONS. THE LANGUAGE HAS A PRIMITIVE OPERATION TO PRUNE ONE BRANCH AT A TIME. THE CONSTRUCTION:

G:F >> V:P

IS A PREDICATE WHICH IS TRUE WHEN F>>P IS. FURTHERMORE WHEN THIS IS THE CASE, A NEW FORMULA G IS CONSTRUCTED CONSISTING OF F WITH THE SUB-EXPRESSION MATCHING P REPLACED BY THE VALUE OF V, AND AS IN THE

PREVIOUS CASE, V IS ASSIGNED THE SUB-EXPRESSION OF F MATCHED BY P. PUT ANOTHER WAY, THERE IS AN EXCHANGE OF VALUES BETWEEN V AND THE SUB-EXPRESSION OF F MATCHED BY P, WHICH THEN BECOMES G.

NOTICE THAT IN THE FIRST CASE THE PATTERN P DESCRIBES THE PROPERTIES OF A SUB-EXPRESSION WE WANT TO FOCUS OUR ATTENTION ON, WHILE IN THE SECOND CASE THE PATTERN P DESCRIBES THE PROPERTIES OF THE IRRELEVANT SUB-EXPRESSIONS WE WANT TO PRUNE.

WHEN THE VALUE OF V IS AN ATOMIC VARIABLE, THE SECOND CASE IS CALLED AN 'ANTI-SUBSTITUTION' SINCE SUB-EXPRESSIONS ARE REPLACED BY NAMES.

THE FOLLOWING EXAMPLE GIVES AN APPLICATION FOR PRUNING FORMULAE. THE ATOMIC VARIABLES WITHIN A FORMULA REPRESENT BOTH INTRINSIC VARIABLES AND PARAMETERS. THE LATTER ARE NOT VARIABLES BUT UNKNOWN CONSTANTS. A SUB-EXPRESSION CONTAINING ONLY CONSTANTS IS ITSELF A CONSTANT. FOR MANY MATHEMATICAL PROCESSES (E.G. DIFFERENTIATION) THE STRUCTURE OF A CONSTANT SUB-EXPRESSION IS IMMATERIAL. BEFORE STARTING ONE OF THOSE MATHEMATICAL PROCESSES WE MAY WANT TO PRUNE ALL THE SUB-EXPRESSIONS CONTAINING ONLY CONSTANTS.

THE FOLLOWING PIECE OF FORMULA ALGOL PROGRAM CONSTRUCTS A FORMULA F FROM A FORMULA G SUCH THAT ALL SUB-EXPRESSIONS OF G NOT CONTAINING AN OCCURRENCE OF THE VARIABLE X ARE REPLACED BY A SUBSCRIPTED VARIABLE NAME: A.[I] THE ARRAY ELEMENTS A[I] WILL CONTAIN THE ACTUAL SUB-EXPRESSIONS.

```
F ← G;  V ← A.[1];
FOR I←1 STEP 1 WHILE F:F >> V:OF(NOTX) DO BEGIN
A[I] ← V;  V ← A.[I+1] END;
```

WHERE THE BOOLEAN PROCEDURE 'NOTX' IS:

```
BOOLEAN PROCEDURE NOTX(S); VALUE S; FORM S;
NOTX ← ¬( (S>>X) v (S==A.[ANY]) );
```

THE VARIOUS USES OF THE PREDICATE '>>' ARE COMBINED IN MANY APPLICATIONS. FOR INSTANCE, ALGORITHMS FOR ALGEBRAIC TRANSFORMATIONS ARE NATURALLY DESCRIBED AS FOLLOWS:

1. LOOK FOR SUB-EXPRESSIONS OF A FORMULA THAT HAVE SOME SPECIFIC CHARACTERISTICS.
2. CONSTRUCT A NEW EXPRESSION USING THE VARIOUS PARTS OF THE SUB-EXPRESSION FOUND.
3. REPLACE THE ORIGINAL SUB-EXPRESSION IN THE FORMULA BY THE NEW ONE.

NOW CONSIDER THE CASE OF TRANSFORMING LOGICAL FORMULAE INTO DISJUNCTIVE NORMAL FORM. WE LOOK FOR SUB-EXPRESSIONS WHICH ARE INSTANCES OF ONE OF THE EXPRESSIONS ON THE LEFT SIDE OF THE NEXT TABLE AFTER FINDING SUCH AN INSTANCE, WE REARRANGE IT, ACCORDING TO THE RULE INDICATED IN THE RIGHT SIDE PART.

¬(A ^ B)	¬A v ¬B
¬(A v B)	¬A ^ ¬B
¬ ¬ A	A
(A v B)^C	A^C v B^C
C^(A v B)	C^A v C^B

THE FOLLOWING PROGRAM MAPS ANY LOGICAL FORMULA INTO A DISJUNCTIVE NORMAL FORM. ALTHOUGH THERE ARE EASIER WAYS TO DESCRIBE THE

SAME ALGORITHM IN FORMULA ALGOL (SEE APPENDIX-6), THIS PROGRAM ILLUSTRATES MANY OF THE POWERFUL FEATURES OF THE LANGUAGE.

```
L1:  BEGIN FORM A, B, C, D, V;
           INTEGER I; SYMBOL P, AND;
           FORM ARRAY T[1:4];
L2:  AND ← /[OPERATOR:^][COMM:TRUE];
L3:  P ← [ ¬(A:ANY ^ B:ANY),
           ¬(A:ANY v B:ANY),
           ¬ ¬ A:ANY,
           (A:ANY v B:ANY) |AND| C:ANY ][INDEX:I];
L4:  T[1] ← ¬A v ¬B;
     T[2] ← ¬A ^ ¬B;
     T[3] ← A;
     T[4] ← AvC ^ BvC;
L5:  V ← .V;
L6:  IF  F:F >> V:OF(P)  THEN BEGIN
L7:  F  ←  SUBS(.V)F( REPLACE(T[I]) );
L8:  GOTO L5 END;
L9:
```

LET US EXAMINE THIS PROGRAM.

STEP L1, DECLARATION OF VARIABLES WITH THEIR CORRESPONDING TYPES.

STEP L2, A CLASS OPERATOR IS DEFINED USING THE SYMBOL VARIABLE 'AND' TO REPRESENT A COMMUTATIVE '^'.

STEP L3, THE SYMBOL VARIABLE P IS ASSIGNED THE LIST OF PATTERNS WHICH CHARACTERIZE THE FOUR POSSIBLE CLASSES OF SUB-EXPRESSIONS THAT WE WANT TO TRANSFORM. THE VALUE OF P ALSO GETS THE ATTRIBUTE 'INDEX' WITH ITS CORRESPONDING VALUE, THE VARIABLE 'I'.

STEP L4, T[1], .., T[4] RECIEVE THE DESCRIPTIONS OF THE TRANSFORMATIONS. SUCH ARRAY ELEMENTS ARE IN A 1-1 CORRESPONDENCE WITH THE PATTERNS STORED IN THE LIST P.

STEP L5, THE VALUE OF V IS SET TO BE THE ATOMIC NAME 'V'.

STEP L6, IF F HAS A SUB-EXPRESSION WHICH IS AN INSTANCE OF ANY OF THE PATTERNS STORED IN P THEN GO TO L7 ELSE GO TO L9. FURTHERMORE IN CASE OF A SUCCESSFUL MATCH, THE VARIABLES A, B, AND POSSIBLY C, WILL BE ASSIGNED THE ACTUAL PARTS OF THE SUB-EXPRESSION THAT HAS MATCHED THE PATTERN. THE ORDINAL POSITION OF THIS PATTERN WITHIN THE LIST P WILL BE THE VALUE OF THE VARIABLE I. F IS ALTERED SO THE MATCHED SUB-EXPRESSION IS REP_ACED BY THE ATOMIC VARIABLE 'V'.

STEP L7, T[I] HAS THE RULE TO BE APPLIED, THUS REP_ACE(T[I]) IS THE ACTUAL TRANSFORMED SUB-EXPRESSION. THE LOCATION OF THE OCCURRENCE OF THE VARIABLE V IN F MARKS THE PLACE WHERE THE SUB-EXPRESSION WAS LOCATED. WE PUT THE REARRANGED SUB-EXPRESSION BACK INTO F BY SUBSTITUTION OF THE VALUE OF REPLACE(T[I]) FOR THE NAME V.

STEP L8, CONTINUE TO LOOK FOR MORE SUB-EXPRESSIONS TO TRANSFORM.

STEP L9, THERE ARE NO MORE SUB-EXPRESSIONS TO TRANSFORM. WHEN NONE OF THE PATTERNS CONTAINED IN P MATCHES ANY SUB-EXPRESSION OF F, IT CAN BE PROVED THAT F IS IN A DISJUNCTIVE NORMAL FORM.

IT WAS PRECISELY THIS PROGRAM THAT POINTED OUT THE NEED FOR A MORE POWERFUL DEVICE TO CARRY OUT TRANSFORMATIONS OF FORMULAE. SUCH A DEVICE WILL BE OUR NEXT SUBJECT OF DISCUSSION.

MARKOV ALGORITHMS.

MOST ALGEBRAIC TRANSFORMATIONS COULD BE PROGRAMED IN FORMULA ALGOL USING AN ALGORITHM ORGANIZATION SIMILAR TO THAT OF THE PRECEDING EXAMPLE. NOTWITHSTANDING THAT THIS PROGRAM ALREADY SHOWS A GREAT DEAL OF THE COMPUTATIONAL POWER IN FORMULA ALGOL, THE LARGE AMOUNT OF PROGRAMMING FOR CONTROL AND BOOKKEEPING IS STILL DISTURBING COMPARED TO THE AMOUNT OF PROGRAMMING FOR DESCRIBING THE 'ESSENCE' OF THE TRANSFORMATION. FURTHERMORE THIS CONTROL PROGRAM WILL, IN AN ALMOST UNCHANGING WAY, OCCUR IN EVERY SUCH TRANSFORMATION. FOR THIS REASON WE INTRODUCED THE 'MARKOV ALGORITHM' FEATURE OF THE LANGUAGE, THE SYNTACTIC CONSTRUCTION OF THIS FEATURE IS BASED ON THE 'NORMAL ALGORITHMS' DEFINED BY A. MARKOV [19].

SOME PROGRAMMING LANGUAGES [7],[8],[10],[13] HAVE USED THE CONCEPT OF NORMAL ALGORITHM AS DEFINED BY MARKOV, BUT AS WE WILL SEE LATER, OUR INTERPRETATION HAS TWO NEW CHARACTERISTICS.

AN ALGEBRAIC TRANSFORMATION IS DESCRIBED BY MEANS OF A LIST OF TRANSFORMATION RULES, CALLED PRODUCTIONS. A PRODUCTION IS A PAIR OF FORMULAS:

P → T

WHERE P IS A PATTERN STRUCTURE AND T IS A FORMULA CALLED THE TRANSFORMAND. THE PATTERN P HAS LABELS WHICH NAME THE DIFFERENT PARTS THAT A FORMULA MUST HAVE IN ORDER TO BE A MEMBER OF THE CLASS DEFINED BY P. THE TRANSFORMAND HAS AS ATOMIC VARIABLES THE NAMES OF THE LABELS USED IN P. THUS IT DESCRIBES THE TRANSFORMATION DESIRED IN A FORMULA WHENEVER

IT HAS THE CHARACTERISTICS DESCRIBED BY P. FOR EXAMPLE,

```
(A:ANY + B:ANY)*C:ANY  →  A*C + B*C
```

IS A PRODUCTION WHICH GIVES THE RULE TO MULTIPLY OUT A FORMULA. THE PRODUCTION'S LEFT SIDE IS A PATTERN THAT DEFINES THE CLASS OF FORMULAE THAT CAN BE MULTIPLIED OUT. THE PRODUCTION'S RIGHT SIDE GIVES THE RULE FOR MULTIPLYING OUT. THE DISTRIBUTIVE LAW OF '+' WITH RESPECT TO '*' GUARANTEES THE CONSERVATION OF ALGEBRAIC EQUIVALENCE.

A LIST OF PRODUCTIONS IS CALLED A SCHEMA AND IT IS STORED IN A SYMBOL VARIABLE, FOR EXAMPLE

```
RATIO ← [  A |PLUS| B/C  →  (.A*.C + .B)/.C  ,
           A - B/C       →  (.A*.C - .B)/.C  ,
           B/C - A       →  (.B - .A*.C)/.C  ,
           A |TIMES| B/C →  (.A*.B)/.C       ,
           (A/B)/C       →  .A/(.B*.C)       ,
           A/(B/C)       →  (.A*.C)/.B       ];
```

THE SYMBOL VARIABLE 'RATIO' IS THE SCHEMA FOR A MARKOV ALGORITHM THAT WILL TRANSFORM ANY FORMULA CONTAINING ONLY THE OPERATORS '+', '-', '*' AND '/' INTO A RATIONAL FORM. I.E., THE SCHEMA WILL CREATE A FORMULA WITH AT MOST ONE OCCURRENCE OF THE OPERATOR '/', AND IF PRESENT, '/' WILL BE THE FORMULA'S MAIN CONNECTOR.

THE OPERTOR '↓' IS USED TO APPLY A SCHEMA TO A FORMULA, FOR EXAMPLE

```
F ↓ RATIO
```

WILL PRODUCE A FORMULA OBTAINED FROM THE APPLICATION OF THE SCHEMA 'RATIO' TO THE FORMULA 'F'.

WE MUST DEFINE WHAT IT MEANS TO APPLY A PRODUCTION TO A FORMULA. WE HAVE A FORMULA WITH MANY BRANCHES WHERE A PRODUCTION COULD BE APPLIED

AND WE HAVE A SCHEMA WITH MANY PRODUCTIONS WE CAN CHOOSE FROM. THERE ARE TWO 'SEQUENCING MODES' EACH OF WHICH DEFINES A WAY TO CHOOSE PRODUCTIONS FROM A SCHEMA TOGETHER WITH A CORRESPONDING WAY TO CHOOSE DIFFERENT BRANCHES OF A FORMULA ON WHICH TO MATCH SUCH PRODUCTIONS.

1.- ONE-BY-ONE SEQUENCING MODE, CORRESPONDS TO A SYNTACTIC CONSTRUCTION OF THE FORM

S ← [P1→T1, P2→T2, ... , PN→TN]

FOR EACH ONE OF THE PRODUCTIONS PJ→TJ, J=1,...,N WE HAVE:

THE PATTERN PJ IS APPLIED TO F. IF THE APPLICATION SUCCEEDS, F IS REARRANGED IN ACCORDANCE WITH TJ AND CONTROL RETURNS TO THE FIRST PRODUCTION, WHICH IS REAPPLIED TO THE RESULT. IF PJ FAILS TO APPLY TO F, IT IS APPLIED RECURSIVELY TO EACH SUB-EXPRESSION OF F, WHERE THE SUB-EXPRESSIONS ARE SOUGHT FROM RIGHT TO LEFT. THEREFORE, PRODUCTION PK IS APPLIED TO F IF AND ONLY IF PRODUCTION PK-1 IS NOT APPLICABLE EITHER TO F ITSELF OR TO ANY SUB-EXPRESSION OF F. THIS SEQUENCING WILL STOP WHEN THE LAST PRODUCTION CANNOT BE APPLIED TO EITHER F OR ANY OF ITS SUB-EXPRESSIONS.

2.- PARALLEL SEQUENCING MODE, CORRESPONDS TO A SYNTACTIC CONSTRUCTION OF THE FORM

S ← [[P1→T1, P2→T2, ..., PN→TN]]

FOR EACH ONE OF THE PRODUCTIONS PJ→TJ, J=1,...,N WE HAVE

THE PATTERN PJ IS APPLIED TO F. IF THE APPLICATION SUCCEEDS, F IS REARRANGED IN ACCORDANCE WITH TJ AND CONTROL RETURNS TO THE FIRST PRODUCTION, WHICH IS REAPPLIED TO THE RESULT. ON THE OTHER HAND, IF THE

APPLICATION OF PJ FAILS, PRODUCTION PJ+1 IS APPLIED TO F. IF PN FAILS AT THE TOPMOST LEVEL OF F, THE WHOLE SEQUENCE IS APPLIED RECURSIVELY TO THE MAIN SUB-EXPRESSIONS OF F, WHERE THE SUB-EXPRESSIONS ARE ALSO SOUGHT FROM RIGHT TO LEFT. THUS, IN PARALLEL SEQUENCING, EACH ONE OF THE PRODUCTIONS IS APPLIED AT LEVEL K OF THE FORMULA ONLY IF ALL PRODUCTIONS HAVE FAILED AT LEVEL K-1. THE TERMINATION CONDITION IS REACHED WHEN ALL PRODUCTIONS FAIL AT THE BOTTOM LEVEL OF EACH SUB-EXPRESSION OF F.

IN GENERAL, A SCHEMA WILL HAVE A COMBINATION OF BOTH SEQUENCING MODES, E.G., A SCHEMA MIGHT BE THE FOLLOWING:

S ← [P1→T1, [P2→T2, P3→T3], P4→T4];

THE PATTERN P1 IS TRIED OUT FIRST, IT USES ONE-BY-ONE SEQUENCING. IF IT FAILS, P2 AND P3 ARE TRIED OUT USING PARALLEL SEQUENCING; FINALLY IF THEY FAIL, P4 IS TRIED OUT USING ONE-BY-ONE SEQUENCING.

THERE ARE TWO BASICALLY NEW CHARACTERISTICS IN THE INTERPRETATION WE HAVE FOR MARKOV ALGORITHMS.

FIRSTLY, THE TRANSFORMATION OPERATE ON TREE STRUCTURES AND NOT ONLY ON STRINGS AS DO MARKOV'S NORMAL ALGORITHMS AND ALL THEIR PROGRAMMING INTERPRETATIONS. THIS FACT HAS CREATED SOME INTERESTING PROBLEMS CONCERNING THE TERMINATION OF ALGORITHMS THAT WE SHALL EXAMINE IN THE NEXT CHAPTER.

SECONDLY, THE PATTERNS USED IN OUR MARKOV ALGORITHMS ARE NOT ONLY SYNTACTIC, BUT ALSO HAVE SEMANTIC DESCRIPTIONS. FOR EXAMPLE, IN CHAPTER V, WHEN WE DISCUSS THE PROOF OF INEQUALITIES, ONE PROGRAM CONTAINS THE PRODUCTION

```
A: OF(POS) < SQRT(B:ANY)    →    .A↑2 < .B
```

THIS PRODUCTION GIVES A RULE TO ELIMINATE A SQUARE ROOT FROM AN INEQUALITY, PROVIDED THE SMALLER SIDE OF THE INEQUALITY IS POSITIVE. THERE IS NO SYNTACTIC CONSTRAINT IN THE FORMULA THAT SHOULD LEAD TO A MATCH OF THE PATTERN LABELED 'A'. WHAT IS REQUIERED IS THAT, IF A FORMULA IS GIVEN TO THE BOOLEAN PROCEDURE 'POS', THIS PROCEDURE MUST PROVE THAT FOR ANY POSSIBLE ASSIGNMENT OF VALUES TO ALL OF THE VARIABLES IN THE FORMULA, THE VALUE OF THE FORMULA WILL BE A POSITIVE NUMBER. FOR EXAMPLE FOR THE FORMULA X↑2+ABS(SIN(Y)) THE BOOLEAN PROCEDURE 'POS' SHOULD YIELD THE VALUE TRUE.

SOME EXAMPLES AND FINAL REMARKS.

HOPEFULLY THE EXAMPLES DESCRIBED SO FAR HAVE GIVEN AN IDEA OF THE FLEXIBILITY THE LANGUAGE HAS WITH WHICH TO ACCOMPLISH ALGEBRAIC MANIPULATIONS. TO CONCLUDE THIS CHAPTER, WE ARE GOING TO COMMENT ON SOME GENERAL CHARACTERISTICS OF THE LANGUAGE AND RELATE THEM WITH WORKING EXAMPLES LISTED AS APPENDICES TO THIS DISSERTATION.

THE LANGUAGE HAS A GREAT DEAL OF FLEXIBILITY TO DESCRIBE ALGORITHMS. THE STRUCTURE OF MOST PROGRAMMING LANGUAGES USALLY FORCES THE USE OF A SINGLE STYLE OF PROGRAMMING; THIS IS NOT THE CASE WITH FORMULA ALGOL. FOR EXAMPLE, CONSIDER THE PROBLEM OF SOLVING AN EQUATION IN X WHEN THE EQUATION CONTAINS ONLY ONE OCCURRENCE OF THE VARIABLE X. THE LANGUAGE'S ITERATIVE STRUCTURE, INHERITED FROM ALGOL, ALLOWS A NATURAL PROGRAMMING FOR AN ITERATIVE ALGORITHM (APPENDIX-3). BUT THE LANGUAGE HAS

ALSO RECURSIVE PROCEDURES; THUS IT IS JUST AS NATURAL TO PROGRAM A RECURSIVE ALGORITHM (APPENDIX-4). YET, THE MARKOV ALGORITHM FEATURE OF THE LANGUAGE PROVIDES A TOOL FOR A THIRD ALGORITHM (APPENDIX-5).

AS WE POINTED OUT EARLIER IN THIS CHAPTER THE MEANING OF A SYNTACTIC CONSTRUCTION IS DETERMINATED BY THE 'TYPE' DECLARATIONS. IN PARTICULAR, WE CAN GIVE TO ALGOL'S ARITHMETIC RELATIONS AND BOOLEAN EXPRESSIONS THE MEANING OF ALGEBRAIC INEQUALITIES AND LOGICAL FORMULAE RESPECTIVELY. THIS FACT PROVIDES THE LANGUAGE WITH A NATURAL NOTATION FOR DESCRIBING GEOMETRIC REGIONS. FOR INSTANCE, WE CAN DESCRIBE REGIONS IN THE REAL LINE AS FOLLOWS: FIRST, WE CHOOSE A FORM VARIABLE, SAY X, TO BE THE REAL LINE VARIABLE. AFTERWARDS, REGIONS COULD BE DEFINED AS A SUB-CLASS OF BOOLEAN EXPRESSIONS:

```
⟨REGION PRIMARY⟩ ::= X ⟨RELATION OP⟩ ⟨EXPRESSION WITH NO X'S⟩
⟨REGION⟩ ::= ⟨REGION PRIMARY⟩ | ⟨REGION⟩ ^ ⟨REGION⟩
             ⟨REGION⟩ ∨ ⟨REGION⟩ | ¬ ⟨REGION⟩
```

FOR EXAMPLE THE LINE SEGMENT (3,4] IS REPRESENTED BY THE FORMULA ALGOL EXPRESSION

$$X > 3 \ \wedge \ X \neg> 4$$

WE HAVE THE FOLLOWING FACTS: THE UNION OF TWO REGIONS R AND S IS DESCRIBED IN FORMULA ALGOL AS R∨S; THEIR INTERSECTION IS DESCRIBED AS R^S AND THE COMPLEMENT OF A REGION R IS DESCRIBED AS ¬R. A POINT Y IS IN A REGION R IF AND ONLY IF EVAL(X)R(Y) IS TRUE. THE EMPTY SET IS REPRESENTED BY 'FALSE'; THE ENTIRE REAL LINE IS REPRESENTED BY 'TRUE'. THE SAME REGION IN THE REAL LINE CAN BE REPRESENTED IN MANY WAYS; ONE SUCH WAY IS AS THE UNION OF SEGMENTS, APPENDIX 6 HAS A PROGRAM TO MAP ANY REGION INTO SUCH REPRESENTATION.

THE ABILITY TO GENERATE NAMES AT EXECUTION TIME FOR A VARIABLE NUMBER OF FUNCTIONS IS FOUND IN FORMULA ALGOL. WE CAN USE DOT-ARRAYS FOR THIS PURPOSE. SUPPOSE ONE HAS A DIFFERENTIAL EQUATION:

$$F(Y^{[K]}, Y^{[K-1]}, \ldots Y^{[1]}, Y, T) = C$$

WHERE Y IS AN UNKNOWN FUNCTION OF THE INDEPENDENT VARIABLE T AND $Y^{[J]}$ IS THE J-TH DERIVATIVE OF Y, WHICH FROM NOW ON WE WILL DENOTE AS 'Y.[J]'. AN APPROXIMATE SOLUTION TO THE DIFFERENTIAL EQUATION IN THE NEIGHBORHOOD T=A IS FOUND BY COMPUTING THE FIRST N TERMS OF THE TAYLOR EXPANSION OF Y. IF THE DIFFERENTIAL EQUATION CONTAINS ONLY ONE OCCURRENCE OF THE K-TH DERIVATIVE WE CAN SOLVE THIS PROBLEM EASILY WITH FORMULA ALGOL. WE DECLARE:

```
FORM ARRAY Y[0:N]
```

AND CONSTRUCT A FORMULA THAT REPRESENTS THE ABOVE DIFFERENTIAL EQUATION. FOR THIS WE USE Y.[J] TO REPRESENT THE J-TH DERIVATIVE, AND Y.[0] TO REPRESENT THE FUNCTION. THE BOUNDARY CONDITIONS OR INITIAL VALUES (T=A) WILL BE STORED IN THE ARRAY ELEMENTS Y[0], ..., Y[K]. FINALLY WE EXTEND THE DERIVATIVE ROUTINE (PROCEDURE 'DERV') TO INCLUDE THE DIFFERENTIATION RULE:

```
IF  F == Y.[J:INTEGER] THEN
DERV ← Y.[J+1] ELSE ...
```

WHICH CLEARLY DESCRIBES THE FACT THAT THE DERIVATIVE OF THE J-TH DERIVATIVE IS THE (J+1)-ST DERIVATIVE. APPENDIX-7 HAS A PROGRAM TO SOLVE A DIFFERENTIAL EQUATION WHICH HAS BEEN REPRESENTED IN THE MANNER HERE DESCRIBED. THE ALGORITHM USES THE FACT THAT THERE IS ONLY ONE OCCURRENCE OF Y.[K], SOLVING THE EQUATION FOR Y.[K] USING ONE OF THE ALGORITHMS

DESCRIBED IN APPENDICES 3, 4 OR 5. AFTERWARDS IT COMPUTES SUCCESSIVELY THE DERIVATIVES Y.[K+1], ..., Y.[N] AND EVALUATES THEM FOR T=A. ONCE THE FIRST N DERIVATIVES ARE COMPUTED THE TAYLOR SERIES APPROXIMATION IS CONSTRUCTED.

TO CONCLUDE, LET US EXAMINE THE 'SIMPLIFICATION' POLICY IN FORMULA ALGOL.

THE SIMPLIFICATION OF FORMULAE HAS A LOCAL RATHER THAN A UNIVERSAL MEANING. WHAT IS SIMPLE IN ONE CONTEXT IS VERY COMPLEX IN ANOTHER. THE SIMPLIFICATION OF A FORMULA IS ALWAYS RELATIVE TO A GOAL. WE SAY THAT A FORMULA HAS BEEN SIMPLIFIED WHEN SUCH A FORMULA HAS BEEN TRANSFORMED INTO ANOTHER THAT MAKES THE GOAL EASIER TO ACHIEVE. FOR THIS REASON WE OPPOSE A BUILT-IN SIMPLIFICATION PACKAGE WHICH WILL OPERATE WITHOUT USER CONTROL. COLLECTING ALIKE TERMS IN A POLYNOMIAL IN GENERAL IS A VERY DESIRABLE THING; BUT IF SUCH A DEVICE ALWAYS OPERATES,ONE COULD NOT, FOR EXAMPLE, POSSIBLY USE THE STANDARD TECHNIQUE TO COMPLETE A PERFECT SQUARE, NAMELY ADDING AND SUBSTRACTING THE SAME TERM. AS FAR AS THE SIMPLIFICATION OF FORMLAE IS CONCERNED, FORMULA ALGOL CAN BE USED AT THREE LEVELS OR, IF THE READER WISHES, IT CAN BE USED BY THREE KINDS OF USERS.

I. BASIC LEVEL.- HERE THE USER CAN CREATE ANY DATA STRUCTURE THAT IS WELL DEFINED IN THE LANGUAGE, HE CAN TEST SUCH DATA STRUCTURES OR BREAK THEM INTO PIECES; HE CAN USE EVAL TO EVALUATE HIS FORMULAE WITH THE STANDARD INTERPRETATION OR HE CAN WRITE HIS OWN EVAL ROUTINE; HE MAY ALSO CREATE HIS OWN OPERATORS (BY MEANS OF DOT-PROCEDURES), HE DEFINES

THE INTERPRETATION OF HIS FORMULAE AS WELL AS THE EVALUATION RULES FOR THEM. THEREFORE, TO BE CONSISTENT WITH THIS LEVEL, THE USER CAN AND MUST DEFINE HIS OWN ALGORITHMS FOR SIMPLIFICATION SINCE HE IS THE ONLY ONE HOW KNOWS THEIR MEANING, E.G., HE MAY USE '+' TO DENOTE GROUP MULTIPLICATION. THE LANGUAGE PROVIDES THE USER AT THIS LEVEL WITH POWERFUL MECHANISMS WHICH ALLOW HIM TO CONSTRUCT SOPHISTICATHED ALGORITHMS.

II. INTERMEDIATE LEVEL.- HERE THE USER HAS THE SAME FREEDOM AS IN THE PREVIOUS LEVEL. BUT INSOFAR AS SIMPLIFICATION OF FORMULAE IS CONCERNED, HE MAY REALIZE THAT MOST OF HIS ALGORITHMS HAVE THE SAME STRUCTURE: THERE ARE BASICALLY A SCANNING OR SEQUENCING THROUGH A FORMULA SEARCHING FOR A A SUB-EXPRESSION THAT MATCHES A GIVEN PATTERN AND WHICH, WHEN FOUND, IS REARRANGED BY THE ALGORITHM. THE PROCESS CONTINUES UNTIL THERE IS NO SUB-EXPRESSION WITH THE CHARACTERISTICS DEFINED BY THE PATTERN OR SET OF PATTERNS. FORMULA ALGOL'S MARKOV ALGORITHMS PROVIDE A RATHER GENERAL MECHANISM FOR DEFINING SIMPLIFICATION ALGORITHMS. THE USER DESCRIBES ONLY THE ESSENCE OF HIS ALGORITHM, BUT HE PAYS A PRICE FOR IT. HE HAS ONLY TWO ALTERNATIVES FOR SEQUENCING THROUGH HIS FORMULAE (IN SOME SIMPLIFICATIONS THE ORDER WHICH DIFFERENT PARTS OF A FORMULA ARE CHANGED MIGHT BE VERY IMPORTANT). FURTHERMORE, HE HAS ONLY ONE ALTERNATIVE FOR CHOOSING THE NEXT RULE TO BE TRIED OUT IN A FORMULA; HE CAN GIVE THE ORDER IN WHICH RULES ARE TO BE APPLIED, BUT THIS ORDER CANNOT CHANGE WHILE THE MARKOV ALGORITHM IS WORKING. IT WILL BE UP TO HIM TO DECIDE IF THE PRICE IS WORTHWHILE. THE IMPORTANT FACT IS THAT HE DOES NOT HAVE TO USE THIS SCHEMA.

III. TOP LEVEL.- HERE AGAIN THE USER HAS ALL THE FREEDOM AS IN

THE PREVIOUS LEVELS BUT BESIDES, HE HAS A LIBRARY OF PROCEDURES WHICH IS PART OF THE SYSTEM BUT NOT PART OF THE LANGUAGE ITSELF. ALL THE STANDARD TECHNIQUES FOR SIMPLIFICATION SHOULD BE IN THIS LIBRARY. APPENDICES 8, 9, 10, 11 AND 12 CONTAIN PROGRAMS TO ELIMINATE UNARY MINUS, TO RATIONALIZE FORMULAE, TO EXPAND FORMULAE BY MULTIPLYING OUT, TO COLLECT ALIKE TERMS IN A POLYNOMIAL AND TO FACTORIZE UP TO SOME DEGREE. THEY WILL BE DISCUSSED IN LATER CHAPTERS WHERE WE SHALL PROVE VARIOUS PROPERTIES OF THESE PROCEDURES.

THUS, A USER CAN WORK AT THE LEVEL THAT MOST FITS HIS NEEDS. IN PARTICULAR, IF HE HAS A STANDARD PROBLEM HE CAN MAKE ALL SIMPLIFICATIONS USING LIBRARY PROCDURES PREVIOUSLY PREPARED. AS FAR AS HE IS CONCERNED, IT MATTERS NOT WHETHER THE PROCEDURES WERE WRITTEN IN MACHINE LANGUAGE OR IN THE LANGUAGE HE IS USING. BUT EVEN MORE, IN THE LATTER CASE WHEN UNEXPECTED BEHAVIOR OCCURS, HE COULD MODIFY THE SIMPLIFICATION PROCEDURE, MAKING A CORRECTION TO SUIT HIS OWN NEEDS. ON THE OTHER HAND, HE CANNOT BE EXPECTED TO CHECK AND CHANGE A SIMPLIFICATION ROUTINE WRITTEN IN

CHAPTER III. ON PROOFS OF ALGORITHMS

FORMULA ALGOL'S MARKOV ALGORITHMS BESIDES PROVIDING A VERY USEFUL PROGRAMMING TOOL MAKE EASIER THE TASK OF PROVING PROPERTIES OF ALGORITHMS. PROVING THAT A PROGRAM TERMINATES USUALLY IS NOT THE PROGRAMER'S CONCERN; THIS IS THE CASE MAINLY BECAUSE IT IS NOT CLEAR IN MOST PROGRAMMING LANGUAGES HOW ONE COULD FORMALIZE THE ANALYSIS TO PROVE FOR ANY INPUT WHITHIN A GIVEN CLASS THAT AN ALGORITHM WILL TERMINATE. BUT FOR A MARKOV ALGORITHM THE PROBLEM IS WELL DEFINED: A MARKOV ALGORITHM 'M' CEASES TO OPERATE IN A FORMULA 'F' WHEN NO PRODUCTION OF 'M' IS APPLICABLE TO EITHER 'F' OR ANY OF ITS BRANCHES. THUS TO PROVE THAT A MARKOV ALGORITHM TERMINATES WE MOST SHOW THAT FOR ANY INPUT FORMULA ONLY FINITE SEQUENCES OF TRANSFORMATIONS CAN BE PRODUCED, I.E., THE LAST FORMULA IN THESE SEQUENCES CANNOT BE FURTHER TRANSFORMED.

MARKOV ALGORITHMS ALSO FACILITATE THE TASK OF CHARACTERIZING THE CLASS OF FORMULAE ON WHICH AN ALGORITHM WILL OPERATE, AS WELL AS THE CLASS OF FORMULAE THAT SUCH AN ALGORITHM WILL PRODUCE.

IN THIS CHAPTER WE ARE GOING TO DEVELOP A BRIEF THEORY THAT YIELDS THREE THEOREMS; THESE THEOREMS GIVE SUFFICIENT CONDITIONS FOR SOME ALGORITHMS TO TERMINATE. SUCH CONDITIONS ARE VERY EASY TO CHECK ON ANY SCHEMA OF PRODUCTIONS BUT IF THE CONDITIONS ARE NOT SATISFIED BY THE SCHEMA THERE IS NOTHING FURTHER WE CAN SAY. BEFORE DEVELOPING ANY THEORY WE ARE GOING TO DISCUSS TWO EXAMPLES THAT HOPEFULLY WILL MOTIVATE BOTH

THE THEOREMS AND THEIR PROOFS. AFTERWARDS WE DEFINE A NUMBER OF CONCEPTS ON WHICH THE THEOREMS AND THEIR PROOFS ARE BASED. WE GIVE EXAMPLES OF NON TERMINATING ALGORITHMS THAT ILLUSTRATE THE NEED FOR SOME CONDITIONS THAT THE THEOREMS REQUIRE. FINALLY WE APPLY THE THEOREMS TO SHOW THE TERMINATION OF SOME ALGORITHMS FOR ALGEBRAIC SIMPLIFICATION AND ALSO WE SHOW THAT THE ALGORITHMS ACTUALLY TRANSFORM ANY MEMBER OF A GIVEN CLASS OF FORMULAE INTO AN ALGEBRAIC EQUIVALENT MEMBER OF OTHER CLASS OF FORMULAE.

TO CHARACTERIZE FORMULAE WE ARE GOING TO USE THE BACKUS NAUR FORM (BNF) [2],[18] NOTATION WITH SOME CHANGES:

1) THE BNF EQUATIONS IN OUR CASE WILL DEFINE FORMULA TREE STRUCTURES RATHER THAN STRINGS OF SYMBOLS. THUS THE PARENTHESES ARE NOT PART OF THE DEFINED OBJECT ITSELF BUT HELP TO INDICATE THE WAY IT IS CONSTRUCTED; OPERATORS ARE THE CONNECTORS IN THE TREE STRUCTURES.

2) A LETTER OR A CONCATENATION OF LETTERS IS AN ABBREVIATION OF A SYNTACTIC CLASS, E.G. PI IS THE SAME AS <PI>.

3) TO ABBREVIATE DEFINITIONS WE INTRODUCE A PARAMETER IN A BNF EQUATION TO DESCRIBE A FAMILY OF DEFINITIONS, FOR EXAMPLE:

$$M[_{10}A]_{10} ::= \text{ATOM} * A \mid M[_{10}A]_{10} / M[_{10}A]_{10} \mid \text{SIN}(M[_{10}A]_{10})$$

DENOTES A FAMILY OF SYNTACTIC DEFINITIONS. EACH DEFINITION WILL DEPEND ON A PARTICULAR VALUE (SYNTACTIC CLASS) ASSIGNED TO THE PARAMETER A. THE DOUBLE CHARACTERS '$[_{10}$' AND '$]_{10}$' ARE METASYNTACTIC BRACKETS USED TO ENCLOSE THE PARAMETER. CONTINUING THE ABOVE EXAMPLE, SUPPOSE THAT T AND U ARE SYNTACTIC CLASSES, SAY

$$T ::= \text{ATOM} \mid T+T \mid T-T$$
$$U ::= \text{ATOM} \mid U*U \mid U/T$$

THEN THE CLASSES R = $M[_{10}T]_{10}$ AND S = $M[_{10}U]_{10}$ WOULD BE

R ::= ATOM * T | R/R | SIN(R)
S ::= ATOM * U | S/S | SIN(S)

APPENDIX A-2 CONTAINS THE BNF DESCRIPTION OF THE FORMULAE WE ARE GOING TO USE IN OUR PROOFS.

WE POINTED OUT IN CHAPTER II THAT WE COULD HAVE MANY ALGORITHMS BASED ON THE SAME SCHEMA OF PRODUCTIONS. THE DIFFERENCE BETWEEN SUCH ALGORITHMS DEPENDS ON THE SEQUENCING RULES THAT GOVERN THE SCHEMA. EACH ALGORITHM IS CHARACTERIZED AS FOLLOWS:

1.- INITIALIZATION. HERE ONE SETS A PRODUCTION P FROM THE SCHEMA AND A BRANCH B FROM THE FORMULA. THIS STEP MAY CHANGE AMONG DIFFERENT SEQUENCING MODES.

2.- TRANSFORMATION. HERE ONE TRIES TO APPLY THE PRODUCTION P TO BRANCH

3.- ITERATION. HERE ONE CHOOSES A PRODUCTION P AND A BRANCH B. THE CHOICE IS BASED ON THE CURRENT STATE, I.E. IT CONSIDERS THE PRODUCTION JUST TRIED, THE BRANCH JUST TRIED AND THE RESULT OF SUCH ATTEMPT. IN THIS STEP LIES THE MAIN DIFFERENCE BETWEEN THE VARIOUS SEQUENCING MODES.

THUS IT IS CLEAR THAT THE SAME SCHEMA OF PRODUCTIONS COULD DEFINE AN ALGORITHM THAT TERMINATES WHEN CERTAIN SEQUENCING RULES ARE UTILIZED AND IT COULD ALSO DEFINE AN ALGORITHM THAT WILL NEVER TERMINATE WHEN OTHER SEQUENCING RULES ARE UTILIZED.

WE ARE INTERESTED IN FINDING A SET OF CONDITIONS THAT GUARANTEE THE TERMINATION OF AN ALGORITHM. THE TERMINATION OF AN ALGORITHM DEPENDS ON THE SCHEMA OF PRODUCTIONS AND ON THE SEQUENCING RULES THAT OPERATE ON SUCH SCHEMA. TO SIMPLIFY OUR PROBLEM WE ARE GOING TO STUDY ONLY

CONDITIONS ON THE SCHEMA OF PRODUCTIONS, I.E., WE ARE GOING TO LOOK FOR CONDITIONS THAT GUARANTEE THE TERMINATION OF ALL ALGORITHMS BASED ON A GIVEN SCHEMA REGARDLESS OF THEIR SEQUENCING RULES.

THUS WE HAVE SEVERAL RELATED DEFINITIONS OF ALGORITHMS. FIRST, THE NORMAL ALGORITHMS (MNA) DEFINED BY A. MARKOV [19]. SECOND, FORMULA ALGOL'S 'MARKOV ALGORITHM' PROGRAMMING DEVICE (FAM). AND THIRD, A COMBINATORIAL TRANSFORMATION SYSTEM (CTS), BASED ON A FORMULA ALGOL SCHEMA OF PRODUCTIONS WITHOUT RESTRICTIONS ON THE ORDER TO CHOOSE PRODUCTIONS FROM THE SCHEMA OR ON THE SELECTION OF SUB-EXPRESSIONS TO BE TRANSFORMED. THE LATTER IS A CONCEPTUAL DEVICE THAT WE SHALL USE TO PROVE TERMINATION THEOREMS. IF A 'CTS' TERMINATES THEN ITS SCHEMA WILL TERMINATE WHEN IT IS OPERATED UNDER THE CONTROL OF ANY FORMULA ALGOL SEQUENCING MODE. THE CONVERSE STATEMENT IS NOT NECESSARILY TRUE.

THE MAIN CHARACTERISTICS OF A 'FAM' ARE:

1. IT OPERATES ON TREE STRUCTURES.

2. THE SUB-EXPRESSIONS TO BE TRANSFORMED ARE CHARACTERIZED BY SYNTACTIC AND/OR SEMANTIC PROPERTIES.

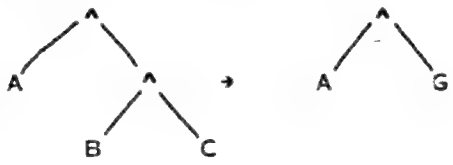
3. IT HAS EXTRACTORS THAT ASSIGN MATCHED BRANCHES TO VARIABLES, THESE VARIABLES CAN THEN BE USED IN ANY MANNER TO DEFINE THE TRANSFORMATION.

4. THERE IS SOME DEGREE OF FREEDOM TO CHOOSE THE ORDER IN WHICH PRODUCTIONS ARE APPLIED.

THUS 'FAM'S ARE A GENERALIZATION OF 'MNA'S, SINCE A STRING TRANSFORMATION CAN BE DESCRIBED BY MEANS OF A TREE TRANSFORMATION. E.G., THE STRING TRANSFORMATION

ABC → AG

CAN BE REPRESENTED AS THE TREE TRANSFORMATION



WHERE '^' IS A CONCATENATION OPERATOR.

IT IS KNOWN THAT THE TERMINATION PROBLEM FOR MARKOV'S NORMAL ALGORITHMS IS UNDECIDABLE. THUS THE TERMINATION PROBLEM FOR FORMULA ALGOL ALGORITHMS (FAM) IS ALSO UNDECIDABLE. THEREFORE THE THEOREMS WE SHALL PROVE CANNOT POSSIBLY GIVE NECESSARY AND SUFFICIENT CONDITIONS FOR THE TERMINATION OF A FAM.

IN ORDER TO MOTIVATE THE DEFINITIONS AND PROOFS NECESSARY TO DEVELOP THE MAIN RESULTS OF THIS CHAPTER WE START BY DISCUSSING TWO SPECIFIC ALGORITHMS AND PROVING SOME STATEMENTS ABOUT THEM.

DEFINITION 1.- THE 'DEPTH' OF AN OCCURRENCE 'O' OF AN OPERATOR IN A FORMULA F IS THE NUMBER OF OPERATORS BETWEEN THE MAIN CONNECTOR OF F AND THE OCCURRENCE O.

DEFINITION 2.- THE 'PATH' OF A PARTICULAR OCCURRENCE 'O' OF AN OPERATOR IN A FORMULA F IS THE LIST OF OPERATORS CONNECTING THE MAIN OPERATOR OF F WITH THE OCCURRENCE O. THUS THE PATH'S LENGTH IS THE DEPTH.

STATEMENT 1.- FOR ANY FORMULA IN CLASS Z THE MARKOV ALGORITHM 'MINUSOUT' (APPENDIX-8) TERMINATES. FURTHERMORE THIS ALGORITHM TRANSFORMS ANY FORMULA IN THE CLASS IG INTO AN ALGEBRAICALLY EQUIVALENT FORMULA WHICH IS A MEMBER OF CLASS TG. THE CLASSES Z, IG AND TG ARE DEFINED IN APPENDIX A-2.

PROOF:

FOR ANY FORMULA F WE DEFINE THE FOLLOWING:

1.- T('-') = DEPTH OF AN OCCURRENCE '-' OF A UNARY MINUS IN F LESS THE NUMBER OF OCCURRENCES OF UNARY MINUS IN THE PATH OF '-'.

B. THE RESULT OF THIS ACTION IS EITHER SUCCESS OR FAILURE. THIS IS A COMMON STEP FOR ALL SEQUENCING MODES.

2.- Q(F) = SUMMATION OF T('-') FOR ALL OCCURRENCES '-' OF UNARY MINUS IN F.

3.- M(F) = NUMBER OF OCURRENCES OF UNARY MINUS IN F.

4.- O(F)=(M(F),Q(F)). O(F) IS GREATER THAN O(G) IF AND ONLY IF M(F)>M(G) OR BOTH M(F)=M(G) AND Q(F)>Q(G).

NOW FOR ANY FORMULA F IN CLASS Z THE PAIR (M(F),Q(F)) IS WELL DEFINED. EVERY TIME A PRODUCTION FROM THE SCHEMA 'MINUSOUT' IS APPLIED TO F EXACTLY ONE OF THE FOLLOWING TAKES PLACE:

1.- M(F) DIMINISHES BY AT LEAST ONE; IF PRODUCTIONS 1, ..., 10 OR 16 ARE APPLIED.

2.- M(F) REMAINS CONSTANT BUT Q(F) DIMISHES BY ONE WHEN PRODUCTIONS 11, ..., 15 ARE APPLIED.

THE ABOVE IS SHOWN BY INSPECTION, E.G. THE PRODUCTION:

$$--A \rightarrow .A$$

INDICATES THAT WHEN IT IS APPLIED TO F TWO OCCURRENCES OF UNARY MINUS WILL BE ELIMINATED. THUS M(F) WILL DIMINISH.

WE HAVE SHOWN THAT EVERY TIME A PRODUCTION IS APPLIED THE PAIR (MF),Q(F)) IS DECREMENTED; BUT M(F) AND Q(F) ARE NON NEGATIVE INTEGERS, THEREFORE WE CAN APPLY THE PRODUCTIONS ONLY A FINITE NUMBER OF TIMES. THIS PROVES TERMINATION.

AFTER THE ALGORITHM HAS TERMINATED NO OCCURRENCE OF UNARY MINUS IS DIRECTLY CONNECTED BELOW AN OCCURRENCE OF '+', '-', '*', '/' OR '↑'. OTHERWISE SOME PRODUCTION COULD BE APPLIED WHICH CONTRADICTS THE FACT THAT THE ALGORITHM HAS TERMINATED. THEREFORE THE OUTPUT IS NO LONGER IN CLASS IG BUT IT IS IN CLASS TG. FINALLY BY SIMPLE INSPECTION WE SEE THAT EACH PRODUCTION IN 'MINUSOUT' PRESSERVES THE ALGEBRAIC EQUIVALENCE, THUS THE OUTPUT IS ALGEBRAICALLY EQUIVALENT TO THE INPUT. QED.

UNFORTUNATELY NOT ALL PROOFS ARE AS SIMPLE AS THE ONE JUST GIVEN. IN FACT IT WAS VERY SURPRISING THAT THE RATHER INTUITIVE PRODUCTION FOR MULTIPLYING OUT:

$$(A + B) * C \rightarrow A * C + B * C$$

TURNED OUT TO BE A VERY DIFFICULT PRODUCTION TO PROVE. I.E., IT IS NOT SIMPLE TO SHOW THAT FOR ANY GIVEN FORMULA F THE ABOVE PRODUCTION CANNOT BE APPLIED INDEFINITELY REGARDLESS OF THE WAY BRANCHES OF F ARE CHOSEN TO BE TRANSFORMED. CONSIDER THE CORRESPONDING TREE TRANSFORMATION:

WE USE THE SAME TECHNIQUE UTILIZED IN THE ABOVE PROOF. NAMELY TO CONSTRUCT A NON NEGATIVE, INTEGER VALUED FUNCTION ASSOCIATED WITH EACH FORMULA AND SUCH THAT EVERY TIME A PRODUCTION IS EXECUTED THE FUNCTION IS DECREMENTED AT LEAST BY ONE. SEVERAL PROBLEMS IMMEDIATELY ARISE. FIRST, THERE IS A PHENOMENON THAT WE CALL THE EXPLOSION PROBLEM: THE BRANCH LABELED C IN THE LEFT SIDE OCURRS TWICE IN THE RIGHT SIDE. THUS MOST

FUNCTIONS BASED ON THE NUMBER OF OCCURRENCES OF SOME OPERATOR IN THE FORMULA COULD INCREASE AS RESULT OF THE TRANSFORMATION. THAT WOULD BE THE CASE WHEN THE BRANCH C HAPPEN TO HAVE OCCURRENCES OF THE OPERATORS CONSIDERED IN THE FUNCTION.

A SECOND PROBLEM IS THAT THE NUMBER OF THOSE OPERATORS IN C HAVE NOT ONLY BEEN MULTIPLIED BY TWO BUT ALSO THAT THEY HAVE INCREASED THEIR DEPTH BY ONE. THUS DECREASING FUNCTIONS BASED ON A NAIVE REDUCTION OF DEPTH ARE ALSO BOUND TO FAIL.

TO PROVE THE TERMINATION OF THE ABOVE PRODUCTION WE CREATE A FUNCTION THAT WE CALL THE 'COMPACTNESS' OF A FORMULA BECAUSE IT DECREASES EVERY TIME SOME FORMULA'S BRANCH GETS EXPANDED. LET US DESCRIBE SUCH A FUNCTION.

GIVEN A FORMULA F, FOR ANY OCCURRENCE OF AN OPERATOR 'OP' IN F WE DEFINE THE FUNCTION PSI('OP') AS FOLLOWS:

$$\text{PSI('OP')} = \begin{cases} 1 & \text{IF 'OP' = '+' OR 'OP' = '-'} \\ \text{PHI(A)} + \text{PHI(B)} & \text{IF 'OP' = '*'} \\ 0 & \text{OTHERWISE} \end{cases} \qquad [1]$$

WHERE PHI IS DEFINED BELOW AND A AND B ARE THE SUB-EXPRESSIONS OF F CONNECTED BY THE OCCURRENCE '*'. NOW LET 'PI' DENOTE THE PATH OF AN OPERATOR IN F, WE DEFINE

$$\text{TAU(PI)} = \text{SUMMATION}_{\text{OP IN PI}}[\ \text{PSI(OP)}\ ] \qquad [2]$$

FOR ANY FORMULA F WE DEFINE:

$$\text{PHI(F)} = \text{MAXIMUM}_{\text{PI IN F}}[\ \text{TAU(PI)}\ ] \qquad [3]$$

FINALLY WE MAKE THE FOLLOWING DEFINITION.

DEFINITION 3.- THE 'COMPACTNESS' OF A FORMULA F IS THE NON NEGATIVE INTEGER VALUED FUNCTION:

$$C(F) = \underset{* \text{ IN } F}{\text{SUMMATION}}[\ 4 \uparrow (PSI('*')*2)\ ] \qquad [4]$$

LET US SHOW THAT THE COMPACTNESS FUNCTION IS WELL DEFINED. IT SUFFICES TO SHOW THAT PSI('OP') IS A WELL DEFINED NON NEGATIVE INTEGER VALUED FUNCTION. THE ONLY QUESTIONABLE CASE IS WHEN 'OP' IS '*'. FROM [1], [2] AND [3] WE HAVE

$$PSI('*') = \underset{PI \text{ IN } A}{\text{MAXIMUM}}[\ \underset{OP \text{ IN } PI}{\text{SUMMATION}}[\ PSI('OP')\ ]\ ] + \underset{PI \text{ IN } B}{\text{MAXIMUM}}[\ \underset{OP \text{ IN } PI}{\text{SUMMATION}}[\ PSI('OP')\ ]\ ] ,$$

I.E., PSI('*') DEPENDS ON THE PSI'S OF OPERATORS BELOW THE OCCURRENCE OF '*'; FOR ANY SUCH OCCURRENCE IN A FORMULA THERE ARE ONLY A FINITE NUMBER OF OPERATORS BELOW IT AND EACH ONE OF THOSE OPERATORS HAS BELOW IT AT LEAST ONE OPERATOR LESS THAN THE NUMBER BELOW THE '*'. THEREFORE THE RECURSION IS WELL DEFINED. QED.

STATEMENT 2.- IF A PRODUCTION (A+B)*C → A*C+B*C IS APPLIED TO ANY BRANCH OF A FORMULA F TO YIELD A FORMULA G THEN THE COMPACTNESS OF G IS SMALLER THAN THE COMPACTNESS OF F.

COROLLARY. FOR ANY FORMULA THE MULTIPLYING OUT PROCESS TERMINATES REGARDLESS OF THE ORDER CHOSEN TO PROCESS DIFFERENT BRANCHES.

PROOF:

LET THE PRODUCTION BE APPLIED TO A SUB-FORMULA Q1 PRODUCING A SUB-FORMULA Q2; LET H BE THE SUPER-FORMULA FROM WHICH Q1 AND Q2 ARE HUNG. I.E., WE HAVE:

[5]

FROM [4] WE SEE THAT WE MUST SHOW:

SUMMATION[4↑[PSI('*')↑2]] > SUMMATION[4↑[PSI('*')↑2]]
* IN F * IN G

FROM [5] WE SEE THAT IT SUFFICES TO SHOW:

1.- THE PSI OF ANY '*' IN A, B OR C DOES NOT CHANGE AS RESULT OF THE TRANSFORMATION.

2.- THE PSI OF ANY '*' IN H DOES NOT INCREASE AS RESULT OF THE TRANSFORMATION.

3.- 4 ↑ (PSI(*1)↑2) > 4 ↑ (PSI(*2)↑2) +

4 ↑ (PSI(*3)↑2) +

SUMMATION[4 ↑ (PSI(*)↑2)]
* IN C [6]

THE PSI'S OF '*' IN A, B, OR C DO NOT CHANGE SINCE THEY DEPEND ONLY ON THE OPERATORS BELOW THE OCCURRENCE '*'. TO SHOW THAT THE PSI'S OF '*' IN H DO NOT INCREASE WE SEE FROM [5] AND [1] THAT IT SUFFICES TO SHOW:

PHI(Q1) ¬< PHI(Q2) [7]

NOW CONSIDER IN GENERAL THE FORMULA 'M' WITH THE STRUCTURE

FROM [3] WE HAVE:

PHI(M) = MAXIMUM[TAU(PI)]
PI IN M

BUT EVERY PATH 'PI' IN M CONTAINS THE OPERATOR 'OP1', THUS FROM [2] AND THE ABOVE WE HAVE:

$$PHI(M) = PSI('OP1') + \underset{\substack{PI\ IN\ X \\ PI\ IN\ Y}}{MAXIMUM}[\ TAU(PI)\ ]$$

$$PHI(M) = PSI('OP1') + MAX[\ PHI(X)\ ,\ PHI(Y)\ ] \qquad [8]$$

WHERE M IS AN ARBITRARY FORMULA WITH MAIN CONNECTOR 'OP1' AND WITH MAIN BRANCHES X AND Y. NOW LET'S COMPUTE PHI(Q1) AND PHI(Q2); FIRST WE DEFINE:

$$RHO = MAX[\ PHI(A)\ ,\ PHI(B)\ ] \qquad [9]$$

FROM [1] AND [5]

$$PSI(*1) = PHI(A+B) + PHI(C)$$

USING [8], [1] AND [9] IN THE ABOVE EQUATION WE HAVE

$$PSI(*1) = 1 + RHO + PHI(C) \qquad [10]$$

THEN FROM [5], [8], [9] AND [10]

$$PHI(Q1) = 1 + RHO + PHI(C) + MAX[\ PHI(C),\ 1+RHO\ ] \qquad [11]$$

ON THE OTHER HAND FROM [5], [1] AND [8]

$$PHI(Q2) = 1 + MAX[\ PHI(B) + PHI(C) + MAX[\ PHI(B)\ ,\ PHI(C)\ ]\ ,\ PHI(A) + PHI(C) + MAX[\ PHI(A)\ ,\ PHI(C)\ ]\ ]$$

$$= 1 + PHI(C) + MAX[\ PHI(B) + MAX[\ PHI(B)\ ,\ PHI(C)\ ]\ ,\ PHI(A) + MAX[\ PHI(A)\ ,\ PHI(C)\ ]\ ]$$

NOW FROM [9]

$$PHI(B) + MAX[\ PHI(B)\ ,\ PHI(C)\ ] \neg\!> RHO + MAX[\ RHO\ ,\ PHI(C)\ ]$$

AND SIMILARLY

$$PHI(A) + MAX[\ PHI(A)\ ,\ PHI(C)\ ] \neg\!> RHO + MAX[\ RHO\ ,\ PHI(C)\ ]$$

FROM THESE INEQUALITIES AND THE VALUE CALCULATED FOR PHI(Q2) WE HAVE

$$PHI(Q2) \neg\!> 1 + PHI(C) + RHO + MAX[\ RHO\ ,\ PHI(C)\ ]$$

WHICH REPLACING [11] PROVES [7]. WE HAVE PROVED THE SECOND POINT IN OUR OUTLINE, NAMELY THE PSI OF '*' IN H DOES NOT INCREASE. NOW LET US PROVE

THE THIRD AND LAST POINT IN THE PROOF'S OUTLINE, NAMELY THE INEQUALITY [6]. FIRST WE SHALL WORK WITH THE TERM PRODUCED BY THE EXTRA OCCURRENCES OF '*' IN C.

$$MU = \text{SUMMATION}[\ 4 \uparrow (PSI('*')\uparrow 2)\ ] \quad * \text{ IN } C \qquad [12]$$

FROM [1] WE SEE THAT AN OCCURRENCE '*' WITHOUT OCCURRENCES OF '+' BELOW IT HAS A PSI EQUAL TO ZERO. THE ONLY OCCURRENCES OF '*' THAT WILL CONTRIBUTE TO MU WOULD BE THOSE WITH +'S BELOW THEM. THE MAXIMUM NUMBER OF '*' (WITH NON ZERO PSI) IN ANY PATH OF C IS AT MOST PHI(C), THUS THE TOTAL NUMBER OF SUCH OCCURRENCES IS LESS THAN 2↑PHI(C). FURTHERMORE THE PSI OF ANY ONE OF THOSE OCCURRENCES OF '*' CANNOT BE GREATER THAN PHI(C); THUS WE HAVE:

$$MU < [2\uparrow PHI(C)] * [4\uparrow(PHI(C)\uparrow 2)]$$

$$MU < [4\uparrow PHI(C)] * [4\uparrow(PHI(C)\uparrow 2)] \qquad [13]$$

FROM [1] AND [5]

$$PSI(*2) = PHI(A) + PHI(C) \qquad [14]$$

$$PSI(*3) = PHI(B) + PHI(C) \qquad [15]$$

DEFINE NU AS FOLLOWS:

$$NU = 4 \uparrow [\ PSI(*2)\uparrow 2\ ] + 4 \uparrow [\ PSI(*3)\uparrow 2\ ] \qquad [16]$$

FROM [14], [15], [16] AND [9]

$$NU \neg> 2 * 4\uparrow[\ (RHO + PHI(C))\uparrow 2\ ] \qquad [17]$$

CASE 1: PHI(C) = 0, THEN

$$MU + NU < 2 * 4\uparrow[\ RHO\uparrow 2\ ]$$

$$< 4\uparrow[\ RHO\uparrow 2 + 1]$$

$$< 4\uparrow[\ (RHO + 1)\uparrow 2\ ]$$

$$= 4\uparrow[\ PSI(*1)\uparrow 2\ ]$$

CASE 2: PHI(C) > 0. THEN FROM [17]

$$NU < 4 \uparrow PHI(C) * 4 \uparrow [(RHO + PHI(C))\uparrow 2]$$

FROM [13] AND ABOVE

$$NU + MU < 4 \uparrow [(RHO + PHI(C))\uparrow 2 + PHI(C) + 1]$$

$$< 4 \uparrow [(RHO + PHI(C) + 1)\uparrow 2]$$

$$= 4 \uparrow [PSI(*1)\uparrow 2]$$

WHICH PROVES [6]. QED.

THIS PROOF SHOWS ONCE MORE THE WELL KNOWN FACT THAT SIMPLE STATEMENTS THAT ARE GENERALLY ACCEPTED ARE QUITE OFTEN CUMBERSOME TO PROVE. ONE OF THE PRODUCTIONS TO REDUCE FRACTIONS, NAMELY

$$A/B + C \rightarrow (A + B*C)/B$$

IS EVEN MORE DIFFICULT TO PROVE. IN TREE NOTATION WE HAVE:

AFTER THE TRANSFORMATION TAKES PLACE THERE ARE TWO COPIES OF B AND THE OPERATORS IN C ARE PUSHED TWO LEVELS DEEPER IN THE TREE STRUCTURE.

RATHER THAN ATTEMPT TO PROVE TERMINATION FOR THIS PRODUCTION BY LOOKING FOR A PARTICULAR FUNCTION THAT DECREASES EVERY TIME THE PRODUCTION IS APPLIED, WE ARE GOING TO DEVELOP A GENERAL THEORY THAT WILL ASSURE NOT ONLY THE TERMINATION OF THIS PRODUCTION BUT OF MANY MORE. WE START BY MAKING SOME DEFINITIONS.

DEFINITION 4.- AN 'ATOMIC ENTITY' IS EITHER A CONSTANT OR A VARIABLE.

DEFINITION 5.- A 'FORMULA' IS EITHER AN ATOMIC ENTITY OR A TREE STRUCTURE WITH OPERATORS USED AS NON TERMINAL NODES AND ATOMIC ENTITIES USED AS TERMINAL NODES.

DEFINITION 6.- TWO FORMULAE A AND B ARE 'IDENTICAL', DENOTED

A :: B

WHENEVER EITHER BOTH ARE THE SAME ATOMIC ENTITY OR IF THEY CAN BE SUPERIMPOSED SUCH THAT THEIR TREE STRUCTURES HAVE A PERFECT MATCH. WHEN THE FORMULAS A AND B ARE NOT IDENTICAL WE WRITE A¬::B.

DEFINITION 7.- A 'SUBSTITUTION' IS A LIST OF ORDERED PAIRS OF FORMULAE ‹X[1]:Y[1], ..., X[N]:Y[N]› DENOTED ALSO ‹X[I]:Y[I];N› WHICH IS RESTRICTED AS FOLLOWS: 1) THE X[I]'S ARE VARIABLES; 2) NO TWO X[I]'S ARE EQUAL; 3) FOR ALL I, X[I] AND Y[I] ARE NOT EQUAL.

SUBSTITUTION IS AN OPERATION THAT MAPS FORMULAE INTO FORMULAE, NAMELY:

LET ‹S› = ‹X[I]:Y[I];N› AND LET F AND H BE FORMULAE, THE IDENTITITY

H :: F ‹S›

INDICATES THAT H IS A FORMULA OBTAINED FROM F BY SUBSTITUTION OF ALL OCCURRENCES OF X[I] IN F FOR Y[I]. ON THE BASIS OF THE ABOVE EQUATION MORE DEFINITIONS ARE MADE:

DEFINITION 8.- THE EXPRESSION F‹S› IS A 'DECOMPOSITION' OF H.

DEFINITION 9.- THE FORMULA F IS A 'GENERALIZATION' OF THE FORMULA H.

DEFINITION 10.- THE FORMULA H IS A 'PARTICULARIZATION' OF THE FORMULA F.

DEFINITION 11.- THE Y[I]'S ARE THE 'COMPLETION VALUES' OF THE FORMULA F RELATIVE TO THE FORMULA H.

DEFINITION 12.- THERE IS AN IMPORTANT PARTICULARIZATION FOR EVERY FORMULA F, CALLED THE 'STANDARDIZATION', AND DEFINED AS FOLLOWS: ‹V[I]:A[I];N›, WHERE V[1], ..., V[N] ARE ALL DISTINCT VARIABLES OF F ARRANGED IN LEXICOGRAPHIC ORDER AND A[1], ..., A[N] ARE THE VARIABLES A1, ..., AN.

THE CONCEPT OF SUBSTITUTION HERE DEFINED IS THE SAME UTILIZED BY THE FORMULA ALGOL LANGUAGE. THE DIFFERENCE IS ONLY IN NOTATION. FORMULA ALGOL'S HAS TWO LISTS: ONE LIST HAS THE VARIABLES X[I] TO BE SUBSTITUTED FOR, AND THE OTHER LIST HAS THE VALUES Y[I] TO REPLACE THE CORRESPONDING VARIABLES GIVEN IN THE FIRST LIST. ON THE OTHER HAND, THE NOTATION HERE INTRODUCED HAS A SINGLE LIST OF PAIRS X[I]:Y[I] EACH ONE DESCRIBING A COMPONENT FOR THE SUBSTITUTION. THIS NEW NOTATION WAS SUGGESTED BY THAT DEVELOPED IN [REF-30], WE CHOOSE THIS NOTATION BECAUSE IT IS MORE CONVENIENT TO FORMALLY DESCRIBE THE STEPS INVOLVED IN A FORMULA ALGOL TRANSFORMATION. FOR EXAMPLE THE FORMULA ALGOL PREDICATE TO TEST WHETHER A FORMULA IS AN INSTANCE OF A PATTERN IS FORMALLY DESCRIBED AS FOLLOWS:

> DEFINITION 13.- A FORMULA F IS 'AN INSTANCE' OF A FORMULA P (DENOTED F==P) WHEN P IS A GENERALIZATION OF F, I.E. WHEN THERE IS A SUBSTITUTION <L> SUCH THAT
>
> F :: P<L>

THE DECOMPOSITION OF F INTO P<L> ALSO FORMALLY DESCRIBES FORMULA ALGOL'S EXTRACTION OPERATIONS. THE COMPLETION VALUES OF P WITH RESPECT TO F ARE PART OF THE SUBSTITUTION <L>, SUCH VALUES ARE CLEARLY THE BRANCHES OF F EXTRACTED DURING THE FORMULA ALGOL PROCESS OF TESTING WHETHER F IS AN INSTANCE OF P. WE CONTINUE MAKING DEFINITIONS:

> DEFINITION 14.- A FORMULA F IS A 'WEAK INSTANCE' OF A FORMULA P, DENOTED
>
> F =W= P
>
> WHENEVER THERE IS A PARTICULARIZATION OF F WHICH IS AN INSTANCE OF P, I.E., IF THERE IS A SUBSTITUTION <L> SUCH THAT
>
> F<L> == P

IN OTHER WORDS THE WEAK INSTANCES OF A FORMULA P CONSTITUTE THE CLASS OF

FORMULAE THAT EITHER ARE ALREADY INSTANCES OF P OR THAT COULD BECOME INSTANCES OF P IF AN APPROPRIATE SUBSTITUTION WERE MADE ON THEM. THIS CLASS IS NOT THE WHOLE CLASS OF FORMULAE SINCE FOR EXAMPLE A+B IS NOT A WEAK INSTANCE OF P*Q BECAUSE NO SUBSTITUTION FOR A AND B ON A+B WOULD MAKE THIS FORMULA AN INSTANCE OF P*Q. NOW LET'S FORMALLY DESCRIBE FORMULA ALGOL'S PREDICATE CONTAIN: '>>'.

DEFINITION 15.- A FORMULA F 'CONTAINS AN INSTANCE' OF P, DENOTED

F >> P

WHENEVER THERE IS A BRANCH OF F THAT IS AN INSTANCE OF P, IN OTHER WORDS, IF AND ONLY IF THERE IS A FORMULA H AND A SUBSTITUTION <L> SUCH THAT

F :: H<L> WHERE <L> IS <X:Y> AND Y==P.

DEFINITION 16.- A 'PRODUCTION' IS AN ORDERED PAIR OF FORMULAE (P→T). THE FIRST FORMULA P IS CALLED THE PATTERN, THE SECOND FORMULA T IS CALLED THE TRANSFORMAND.

DEFINITION 17.- A PRODUCTION (P→T) IS SAID TO BE 'APPLICABLE' TO A FORMULA F WHENEVER F>>P.

DEFINITION 18.- A FORMULA F[0] IS 'TRANSFORMED' INTO A FORMULA F[1], DENOTED F[0] →→ F[1], BY A PRODUCTION (P→T) WHEN THE FOLLOWING TAKES PLACE: FIRST, (P→T) MUST BE APPLICABLE TO F[0], I.E., F[0]>>P. THUS THERE IS A FORMULA H AND SUBSTITUTIONS <S> AND <E> SUCH THAT

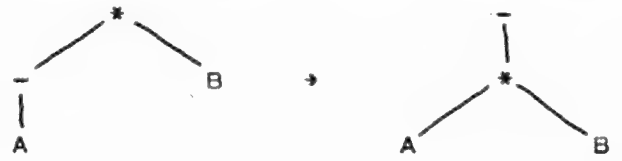
F[0] :: H<S> , WHERE <S> = < X:P<E> >

X OCCURRING ONLY ONCE IN H. F[1] IS CONSTRUCTED BY THE SUBSTITUTION <L>.

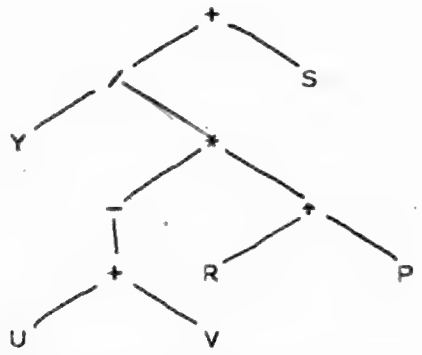
F[1] :: H<L> , WHERE <L> = < X:T<E> >

IN TREE NOTATION WE HAVE:

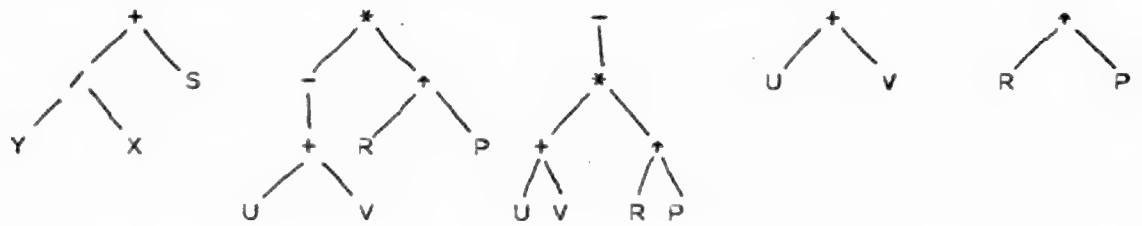
F[0] HAS AN INSTANCE OF P, NAMELY Q0; Q0 IS DECOMPOSED AS P<E>, I.E. <E> HAS THE COMPLETION VALUES (Y[I]'S) OF P RELATIVE TO Q0. F[0] IS ITSELF DECOMPOSED AS H<S>. I.E. H IS THE STRUCTURE OF F[0] THAT REMAINS UNCHANGED AND <S> CONTAINS THE INSTANCE OF P THAT WILL BE TRANSFORMED BY THE PRODUCTION. THE BRANCH Q0 IS REARRANGED TO PRODUCE BRANCH Q1, THE REARRANGMENT IS DEFINED BY THE TRANSFORMAND T WITH THE EXTRACTED VALUES FROM Q0, THUS Q1 IS DECOMPOSED AS T<E>. THE NEW FORMULA F[1] IS OBTAINED BY REPLACING THE BRANCH Q0 BY BRANCH Q1. THUS F[1] IS H<L> WHERE <_> CONTAINS THE NEW BRANCH Q1. FOR EXAMPLE, LET (P→T) BE



AND LET F[0] BE



THEN H, P<E>, T<E>, Y[I] AND Y[2] ARE RESPECTIVELY:



DEFINITION 19.- TWO FORMULAE A AND B ARE 'ESSENTIALLY DIFFERENT' IF AND ONLY IF THE STANDARDIZATION OF A IS NOT IDENTICAL TO THE

STANDARDIZATION OF B.

DEFINITION 20.- A FORMULA G IS A 'NON-TRIVIAL GENERALIZATION' OF A FORMULA F IF AND ONLY IF G IS A GENERALIZATION OF F AND THEY ARE ESSENTIALLY DIFFERENT FORMULAE.

DEFINITION 21.- A FORMULA G IS AN 'INCOMPLETE GENERALIZATION' OF A FORMULA F IF AND ONLY IF G IS A NON-TRIVIAL GENERALIZATION OF F AND FURTHERMORE G HAS NON-TRIVIAL GENERALIZATIONS OF ITS OWN.

DEFINITION 22.- THE 'COMPLETE GENERALIZATION' OF A FORMULA F IS THE FIRST VARIABLE IN THE LEXICOGRAPHIC ORDERING, A1.

DEFINITION 23.- THE 'GENERALIZATION SET' OF A FORMULA IS THE SET OF ALL ESSENTIALLY DIFFERENT INCOMPLETE GENERALIZATIONS OF THAT FORMULA. IT IS, OF COURSE, FINITE.

FOR EXAMPLE, A+B*C, A-B*C, AND X+Y*Y ARE ESSENTIALLY DIFFERENT. A+B*C AND X+Y*Z ARE ESSENTIALLY THE SAME. A/B+X IS A NON-TRIVIAL GENERALIZATION OF A/B+C*D. X+Y IS A TRIVIAL GENERALIZATION OF A+B. A/B+X IS AN INCOMPLETE GENERALIZATION OF A/B+C*D. THE FORMULAE A1+A2, A1/A2+A3, A1+A2*A3 ARE THE MEMBERS OF THE GENERALIZATION SET OF A/A+C*D.

LET US CONSIDER A PRODUCTION (P→T). WE REQUIRE THAT P SHOULD HAVE ONLY ONE OCCURRENCE OF EACH OF ITS VARIABLES. WE REQUIRE A FURTHER SET OF DEFINITIONS.

DEFINITION 24.- A 'VIRTUAL PATTERN' OF A PRODUCTION (P→T) IS A MEMBER OF THE GENERALIZATION SET OF P.

DEFINITION 25.- THE 'SET OF COMPLEMENTARY PATTERNS' OF A PRODUCTION (P→T) IS THE SET DEFINED AS FOLLOWS:

$$C = C1 \vee C2$$

WHERE

$C1 = [\ X \mid X$ IS A COMPLETION VALUE OF H WITH RESPECT TO P$]$

H BEING ANY MEMBER OF THE SET V, AND

$C2 = [\ X \mid X$ IS AN INCOMPLETE GENERALIZATION OF Y$]$

Y BEING ANY MEMBER OF THE SET C1.

DEFINITION 26.- A 'REAL INSTANCE' IS AN INSTANCE OF P.

DEFINITION 27.- A 'VIRTUAL INSTANCE' IS AN INSTANCE OF A GENERALIZATION OF P.

DEFINITION 28.- A 'COMPLEMENTARY INSTANCE' IS AN INSTANCE OF A MEMBER OF C.

DEFINITION 29.- A 'GENERALIZED INSTANCE' OR SIMPLY AN 'INSTANCE' IS EITHER A REAL INSTANCE, A VIRTUAL INSTANCE, OR A COMPLEMENTARY INSTANCE.

DEFINITION 30.- AN INSTANCE A IS 'STRICTLY CONTAINED' IN ANOTHER INSTANCE B IF AND ONLY IF THE MAIN CONNECTOR OF A IS IN B BUT IS NOT THE MAIN CONNECTOR OF B.

DEFINITION 31.- LET X[1], ..., X[N] BE ALL DISTINCT VARIABLES OF P. WE ARE ASSUMMING THAT X[I] OCCURS ONLY ONCE IN P. LET M[I] BE DEFINED AS FOLLOWS:

$$M[I] = \begin{cases} K, & \text{IF } X[I] \text{ OCCURS } K+1 \text{ TIMES IN T} \\ 0, & \text{OTHERWISE} \end{cases}$$

$$M = \sum_{I=1}^{N} M[I]$$

R = TOTAL NUMBER OF BOTH VIRTUAL INSTANCES AND COMPLEMENTARY INSTANCES IN T.

DEFINITION 32.- WE SAY THAT A TRANSFORMAND HAS MULTIPLICITY WHEN $M \neq 0$ AND THAT A TRANSFORMAND HAS REPRODUCTIVITY WHEN $R \neq 0$.

DEFINITION 33.- A PRODUCTION (P→T) IS COMPATIBLE IF AND ONLY IF

1.- THE TRANSFORMAND T DOES NOT CONTAIN A REAL INSTANCE OF P, I.E., $\neg(T >> P)$.
2.- IF THE TRANSFORMAND T CONTAINS VIRTUAL INSTANCES THEY ARE RESTRICTED AS FOLLOWS:
2.1 VIRTUAL INSTANCES IN T CANNOT CONTAIN OTHER INSTANCES.
2.2 MULTIPLICITY CANNOT OCCUR WITHIN A VIRTUAL INSTANCE OF T; THE TRANSFORMAND AS A WHOLE CAN HAVE OF COURSE MULTIPLICITY BUT REPETITIONS OF VARIABLES IN T CANNOT BE LOCATED IN THE SAME SUB-STRUCTURE THAT DEFINES A VIRTUAL INSTANCE.

FINALLY WE SHALL STATE THE FIRST IMPORTANT RESULT OF THIS CHAPTER:

THEOREM I.- IF A PRODUCTION (P→T) IS COMPATIBLE, THEN FOR ANY FORMULA F[0] THERE EXISTS AN INTEGER N¬<0 SUCH THAT F[0] →→ F[1] →→ ... →→ F[N], AND (P→T) IS NOT APPLICABLE TO F[N]. THE LENGTH N OF SUCH SEQUENCES DEPENDS ON THE CHOICE OF THE REAL INSTANCE IN F[I] TO BE TRANSFORMED TO PRODUCE F[I+1].

PROOF:

FROM NOW ON, THE WORD 'INSTANCE' WILL HAVE THE TECHNICAL MEANING GIVEN BY DEFINITION 29.

DEFINITION 34.- THE 'CHARACTERISTIC FUNCTION' PHI(F) OF A FORMULA F RELATIVE TO A PRODUCTION (P→T) IS:

$$PHI(F) = TAU(\ F,\ K(F)\ ) \qquad [18]$$

WHERE

$$K(F) = \text{NUMBER OF STRICTLY CONTAINED INSTANCES IN F.} \qquad [19]$$

TWO STRICTLY CONTAINED INSTANCES (DEF'S-29,30) DO NOT COUNT AS DIFFERENT INSTANCES IF THEY HAVE THEIR MAIN CONNECTOR IN COMMON. IN CASE THEY DO, THE INSTANCE WITH LARGER STRUCTURE WILL BE CONSIDERED. THE FUNCTION TAU IS DEFINED AS FOLLOWS:

$$TAU(\ F,\ N\ ) = \begin{cases} 0 & \text{IF } N=0 \\ \sum_{J=1}^{N} [\ PSI(\ I[F,J]\ )\ ] & \text{IF } N>0 \end{cases} \qquad [20]$$

WHERE I[F,J] IS THE J-TH STRICTLY CONTAINED INSTANCE OF F. INSTANCES MAY BE ORDERED IN MANY WAYS, E.G. FROM TOP TO BOTTOM TAKING LEFT BRANCHES FIRST. THE FUNCTION PSI(I) IS DEFINED AS FOLLOWS:

$$PSI(I) = \begin{cases} MU(\ TAU(\ I,\ K(I)\ )) & \text{IF I IS EITHER A REAL OR A VIRTUAL INSTANCE} \\ 1 & \text{IF I IS A COMPLEMENTARY INSTANCE} \end{cases} \qquad [21]$$

WITH MU DEFINED AS FOLLOWS:

$$MU(X) = (M*X + R + 1)\uparrow X \qquad [22]$$

WHERE M IS THE MULTIPLICITY OF (P→T) AND R IS THE REPRODUCTIVITY OF (P→T). DEFINITION-32.

LEMMA 1.- FOR ANY FORMULA F, PHI(F) IS A WELL DEFINED NON NEGATIVE INTEGER.

LEMMA 2.- F ↠ G IMPLIES PHI(F) > PHI(G).

PROOF OF THEOREM:

BY THE ABOVE LEMMAS ANY SEQUENCE F[0] ↠ F[1] ↠ ... ↠ F[N] IS EQUIVALENT TO A SEQUENCE OF STRICTLY DECREASING NON NEGATIVE INTEGERS. THUS THE SEQUENCE IS FINITE.

PROOF OF THE LEMMA 1:

PROOF BY INDUCTION ON N=K(Y), Y BEING ANY FORMULA.

BASIS.

Y HAS NO INSTANCES. THUS FROM [19] K(Y)=0 AND THEN FROM [18] AND [20] PHI(Y)=TAU(Y,0)=0. THE LEMMA IS TRUE FOR ANY FORMULA WITHOUT INSTANCES.

INDUCTION STEP.

SUPPOSE THE LEMMA IS TRUE FOR K(Y) ¬> N. LET Y BE WITH K(Y)=N+1. FROM [18] AND [20] WE HAVE:

$$PHI(Y) = TAU(\ Y,\ N+1\ )$$

$$= \sum_{J=1}^{N+1} [\ PSI(\ I[Y,J]\ )\ ]$$

IF I[Y,J] IS A COMPLEMENTARY INSTANCE, BY [21] PSI(I[Y,J])=1. SUPPOSE OTHERWISE. SINCE I[Y,J] IS STRICTLY CONTAINED IN Y IT SHOULD HAVE AT LEAST ONE INSTANCE (STRICTLY CONTAINED) LESS THAN Y, NAMELY ITSELF. THEN

K(I[Y,J]) ¬> N. BY THE INDUCTION HYPOTHESIS TAU(I[Y,J],K(I[Y,J])) IS A WELL DEFINED NON NEGATIVE INTEGER THUS FROM [21] AND [22] SO IS PSI(I[Y,J]) AS WILL BE THE SUM OF SUCH PSI'S. QED.

PROOF LEMMA 2

WE MUST SHOW THAT:

PHI(F) > PHI(G)

I.E. FROM [18] AND [20] WE MUST SHOW:

SUMMATION[PSI(I[F,J])] > SUMMATION[PSI(I[G,K])] [23]

THE LEFT SUMMATION IS CARRIED OVER ALL STRICTLY CONTAINED INSTANCES I[F,J] OF F AND THE RIGHT SUMMATION IS CARRIED OVER ALL STRICTLY CONTAINED INSTANCES I[G,K] OF G. THUS WE WANT TO INVESTIGATE THE EFFECT THAT THE TRANSFORMATION F↠G PRODUCES ON THE INSTANCES OF BOTH FORMULAE; IN PARTICULAR WE WANT TO KNOW WHICH INSTANCES ARE ELIMINATED AND WHICH INSTANCES ARE CREATED BECAUSE OF THE TRANSFORMATION. ALSO FOR THOSE INSTANCES THAT ARE PRESENT IN BOTH FORMULAE WE WANT TO KNOW IF THEIR PSI HAS CHANGED. NOW CONSIDER THE TRANSFORMATION F↠G, FROM DEFINITION-18 WE HAVE:

F :: H<S> , <S> = < X:P<E> > [24]

G :: H<L> , <L> = < X:T<E> > [25]

WHERE H HAS ONLY ONE OCCURRENCE OF THE VARIABLE X. THE SUBSTITUTION <E> IS

<E> = < X[I]:Y[I] >

FROM THESE RELATIONS AND WITH THE HELP OF THE TREE STRUCTRE TRANSFORMATION DESCRIBED IN DEFINITION-18 WE CAN CONVINCE OURSELVES THAT THE INSTANCES OF F ARE OF THREE TYPES ONLY.

I. INSTANCES CONTAINED IN H.

II. INSTANCES CONTAINING ONE PART IN H AND THE OTHER PART IN P<E>.

III. INSTANCES CONTAINED IN P<E>, INCLUDING P<E> ITSELF.

SIMILARLY FOR G THERE ARE THREE TYPES OF INSTANCES:

I. INSTANCES CONTAINED IN H.

II. INSTANCES CONTAINING ONE PART IN H AND THE OTHER PART IN T<E>.

III. INSTANCES CONTAINED IN T<E>, INCLUDING (POSSIBLY) T<E>.

NOW FROM [22] WE SEE THAT FOR ANY NON NEGATIVE INTEGER X,

$$MU(X) \not< 1$$

THEN FROM [21] AND THE ABOVE INEQUALITY WE HAVE THAT FOR ANY INSTANCE L

$$PSI(L) \not< 1 \qquad [26]$$

THEN TO PROVE [23] IT SUFFICES TO SHOW:

$$\sum_{\text{TYPE I}} PSI(I[F,J]) > \sum_{\substack{\text{TYPE I} \\ \text{TYPE II}}} PSI(I[G,K]) \qquad [27]$$

AND

$$\sum_{\text{TYPE III}} PSI(I[F,J]) > \sum_{\text{TYPE III}} PSI(I[G,K]) \qquad [28]$$

NOW LET US MARK IN A 1-1 FASHION ALL INSTANCES OF TYPE III WITHIN G THAT ARE IDENTICAL TO INSTANCES OF TYPE III WITHIN F. LET I[NEW] DENOTE ANY INSTANCE OF TYPE III WITHIN G THAT WAS NOT MARKED BY THIS PROCESS. FURTHERMORE FROM [21] AND [20] WE SEE THAT THE VALUE OF PSI(L) DEPENDS ONLY ON THE STRUCTURE BELOW L. THUS BY [26] WE SEE THAT TO PROVE [28] IT SUFFICES TO SHOW

PSI(P<E>) > SUMMATION[PSI(I[NEW])] [29]

LET US POSTPONE FOR A MOMENT THE PROOF OF THIS INEQUALITY AND LET US EXAMINE [27].

THERE IS A 1-1 CORRESPONDENCE BETWEEN THE INSTANCES OF TYPE I WITHIN F AND THE UNION OF INSTANCES OF TYPE I AND TYPE II WITHIN G. INDEED WE HAVE THAT NONE OF THE INSTANCES OF TYPE I WITHIN G WAS CREATED BY THE TRANSFORMATION. THUS THEY WERE BEFORE INSTANCES OF TYPE I WITHIN F. AN INSTANCE OF TYPE II IN G, SAY Q, HAS A GENERALIZATION OF IT, SAY Q', WITHIN H AND THE COMPLETION VALUES OF Q' RELATIVE TO Q ARE IN T<E>, BUT FROM DEFINITIONS 25 AND 26, Q' IS ITSELF AN INSTANCE. THUS FOR ANY INSTANCE OF TYPE II WITHIN G THERE IS A CORRESPONDING INSTANCE OF TYPE I WITHIN F; BUT EVERY INSTANCE OF TYPE I OR TYPE II HAS ITS MAIN CONNECTOR WITHIN H, THE STRUCTURE OF H IS NOT CHANGED BY THE TRANSFORMATION THUS THE INSTANCES ARE IN A 1-1 CORRESPONDENCE.

NOW LOOKING AT [27] WE REALIZE THAT NOTWITHSTANDING THERE IS A 1-1 CORRESPONDENCE BETWEEN INSTANCES OF TYPE I IN F AND INSTANCES OF TYPE I AND TYPE II WITHIN G, YET THEIR PSI'S MIGHT HAVE CHANGED AND INDEED THEY HAVE CHANGED SINCE THEY DEPEND ON THE STRUCTURE BELOW AND SUCH STRUCTURE WAS PRECISELY THE PART THE TRANSFORMATION OPERATED ON. BUT FROM THE DEFINITION OF PSI WE SEE THAT THE PROOF OF [29] WILL IMPLY ALSO THE VALIDITY OF [27]. THEREFORE TO COMPLETE THE PROOF OF THE THEOREM IT REMAINS TO SHOW [29]:

PSI(P<E>) > SUMMATION[PSI(I[NEW])]

THE REAL INSTANCE P<E> IS REPLACED BY T<E>. NEW INSTANCES I[NEW] ARE THOSE INSTANCES IN T<E> THAT ARE NOT IN P<E>. THERE ARE ONLY TWO WAYS

TO CREATE THESE INSTANCES:

1.- BY MULTIPLICITY (THE EXPLOSION PROBLEM), INSTANCES CONTAINED IN Y[I] ARE REPEATED M[I] TIMES WHENEVER X[I] OCCURRS M[I]+1 TIMES IN T. (C.F. DEFINITION 32)

2.- BY REPRODUCTIVITY, THE TRANSFORMAND T ITSELF CONTAINS INSTANCES WHICH MAY BE ONLY VIRTUAL OR COMPLEMENTARY. BY HYPOTHESIS (COMPATIBLE PRODUCTION) WE HAVE THAT ¬(T>>P) THUS THERE ARE NO NEW REAL INSTANCES COMPLETELY CONTAINED IN T.

IF THERE WERE INCOMPLETE INSTANCES IN T THAT THE COMPLETION VALUES DEFINED IN <E> COULD COMPLETE, SUCH INCOMPLETE INSTANCES FROM DEFINITIONS 24 AND 25 WOULD BE INSTANCES THEMSELVES THUS THEY WOULD BE ALREADY CONSIDERED BY CASE 2 ABOVE; THEREFORE THE ABOVE TWO CASES COVER ALL POSSIBLE WAYS TO CREATE NEW INSTANCES.

FROM [18], [19], [20] AND [26] WE HAVE THAT

$$PHI(F) \quad \neg< \quad K(F) \qquad [30]$$

AND IN PARTICULAR WE HAVE

$$PHI(\ P<E>\ ) \quad \neg< \quad K(\ P<E>\ ) \qquad [31]$$

THEN THE NUMBER OF INSTANCES PRODUCED BY MULTIPLICITY IS NOT LARGER THAN

$$M * PHI(\ P<E>\ ) \qquad [32]$$

ANY ONE OF THESE INSTANCES PRODUCED BY MULTIPLICITY, SAY INSTANCE Q, IS A COPY OF AN INSTANCE Q' CONTAINED IN P<E> AND MORE PRECISELY IT IS CONTAINED IN ONE OF THE EXTRACTED VALUES Y[I]. THE PSI OF AN INSTANCE DEPENDS ONLY ON THE STRUCTURE BELOW IT, THUS

$$PSI(Q) \quad = \quad PSI(Q') \qquad [33]$$

BUT Q' IS AN INSTANCE STRICTLY CONTAINED IN P<E>, THEN Q' CONTAINS AT LEAST ONE INSTANCE LESS THAT P<E>. THIS MEANS THAT WHEN THE SUMMATION [20] IS COMPUTED FOR Q' IT HAS AT LEAST ONE TERM LESS THAN WHEN IS

COMPUTED FOR P<E>; THEREFORE BY [26]

$$PHI(Q') \neg> PHI(P<E>) - 1 \qquad [34]$$

NOW REPLACING [18] IN THE ABOVE INEQUALITY

$$TAU(Q', K(Q')) \neg> PHI(P<E>) - 1$$

WE SEE FROM [22] THAT MU(X) IS MONOTONIC INCREASING; APPLYING THIS FACT TO THE ABOVE INEQUALITY AND [21], AND USING [33] WE HAVE:

$$PSI(Q) \neg> [M*(PHI(P<E>)-1) + R + 1]\uparrow[PHI(P<E>)-1]$$

$$PSI(Q) < [M*PHI(P<E>) + R + 1]\uparrow[PHI(P<E>)-1] \qquad [35]$$

THIS LAST INEQUALITY IS TRUE FOR NEW INSTANCES PRODUCED BY MULTIPLICITY. WE SHALL PROVE THAT IT IS TRUE ALSO FOR ALL OTHER NEW INSTANCES. I.E. FOR INSTANCES PRODUCED BY REPRODUCTIVITY.

THE NUMBER OF NEW INSTANCES PRODUCED BY REPRODUCTIVITY IS FIXED, NAMELY

$$R \qquad [36]$$

IF R IS ZERO WE SKIP THE FOLLOWING DISCUSSION UP TO THE SENTENCE BEFORE EQUATION [38]. NOW IF R IS NOT ZERO IT MEANS THAT THE TRANSFORMAND T CONTAINS VIRTUAL AND/OR COMPLEMENTARY INSTANCES, THEREFORE THE PATTERN P CONTAINS INSTANCES SINCE P CAN BE DECOMPOSED AS G<Z> WHERE G IS A VIRTUAL PATTERN AND THE TRANSFORMATION <Z> CONTAINS COMPLEMENTARY PATTERNS, THUS FROM [26], [22] AND [21] WE HAVE

$$PHI(P<E>) > 1 \qquad [37]$$

REPRODUCTIVITY INSTANCES ARE EITHER VIRTUAL OR COMPLEMENTARY. A COMPLEMENTARY INSTANCE HAS A PSI=1 (C.F. EQUATION 21) THEREFORE FROM THIS FACT AND [37] WE SEE THAT ANY COMPLEMENTARY INSTANCE PRODUCED BY REPRODUCTIVITY SATISFIES INEQUALITY [35]. NOW CONSIDER A VIRTUAL

INSTANCE Q PRODUCED BY REPRODUCTIVITY. LET Q BE DECOMPOSED AS

$$Q :: Q'\langle E\rangle$$

I.E. THE PATTERN THAT DEFINES Q IS Q' AND THIS PATTERN IS PART OF THE STRUCTURE OF THE TRANSFORMAND T. BY HYPOTHESIS THE PRODUCTION IS COMPATIBLE (DEFINITION-33); THUS FROM THE DEFINITION'S CONDITIONS 2.1 AND 2.2 WE HAVE THAT Q' HAS NO OTHER INSTANCES AND THAT IT HAS NO MULTIPLICITY; THEREFORE THE INSTANCE Q CONTAINS NO MORE INSTANCES THAN P<E> AND IN FACT IT CONTAINS AT LEAST ONE INSTANCE LESS THAN P<E>, NAMELY ONE OF THE COMPLEMENTARY INSTANCES DEFINED BY THE COMPLETION VALUES OF Q' RELATIVE TO P. WE HAVE SHOWN THAT ANY VIRTUAL INSTANCE PRODUCED BY REPRODUCTIVITY HAS AT LEAST ONE STRICTLY CONTAINED INSTANCE LESS THAN P<E>, THEREFORE THE PARAGRAPH PRECEDING EQUATION [34] IS VALID IN THE CASE OF NEW VIRTUAL INSTANCES AND SO WILL BE [34] AND [35]. WE HAVE PROVED THAT [35] IS VALID FOR ALL NEW INSTANCES.

NOW FROM [32] AND [36] WE HAVE THAT THE TOTAL NUMBER OF NEW INSTANCES IS NOT GREATER THAN

$$M*PHI(P\langle E\rangle) + R \qquad [38]$$

THUS FROM [35] AND [38] WE HAVE

$$\begin{aligned} SUMMATION[PSI(I[NEW])] &< [M*PHI(P\langle E\rangle)+R]*[M*PHI(P\langle E\rangle)+R+1]+[PHI(P\langle E\rangle)-1] \\ &< [M*PHI(P\langle E\rangle) + R + 1]+[PHI(P\langle E\rangle)] \\ &= PSI(P\langle E\rangle) \end{aligned}$$

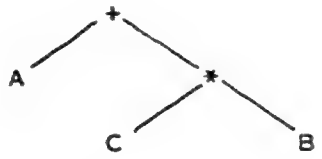
WHICH PROVES [29]. QED.

NOW LET US APPLY THIS THEOREM. CONSIDER THE EXAMPLE MENTIONED EARLIER IN THIS CHAPTER: THE TERMINATION OF A PRODUCTION CONTAINED IN THE ALGORITHM TO REDUCE FRACTIONS.

> STATEMENT 3.- THE PRODUCTION (A/B+C) → (A+C*B)/B CANNOT BE APPLIED INDEFINITLY TO ANY GIVEN FORMULA REGARDELESS OF THE ORDER CHOSEN TO TRANSFORM DIFFERENT PARTS OF SUCH FORMULA.

PROOF:

THE ABOVE PRODUCTION DEFINES ONLY ONE VIRTUAL PATTERN: + THEREFORE THE PATTERN / IS A COMPLEMENTARY PATTERN. NOW THE TRANSFORMAND CONTAINS A VIRTUAL INSTANCE:



BUT IT HAS NO MULTIPLICITY AND CONTAINS NO OTHER VIRTUAL INSTANCE. THE CONDITIONS OF THEOREM I ARE FULFILLED. QED.

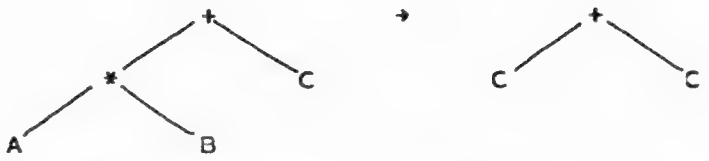
LET US ANALYZE FOR A MOMENT THE CONDITIONS OF THE THEOREM I. THE REQUIREMENT:

$$\neg(\ T\ >>\ P\ )$$

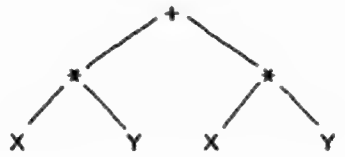
IS CLEARLY A NECESSARY CONDITION SINCE IF THE TRANSFORMAND T CONTAINS AN INSTANCE OF P THEN THE PROCESS WILL NEVER TERMINATE WHEN APPLIED TO A FORMULA THAT CONTAINS AN INSTANCE OF P TO START WITH.

NOW CONSIDER THE REQUIREMENT THAT VIRTUAL INSTANCES CONTAINED IN T SHOULD NOT HAVE MULTIPLICITY. THIS CONDITION WAS CREATED TO ACHIEVE THE PROOF. IT IS NOT KNOWN IF IT IS NECESSARY, BUT THE FOLLOWING EXAMPLE SHOWS A CASE WHICH FAILS TO STOP AND HAPPENS TO VIOLATE THAT REQUIREMENT.

THE PRODUCTION

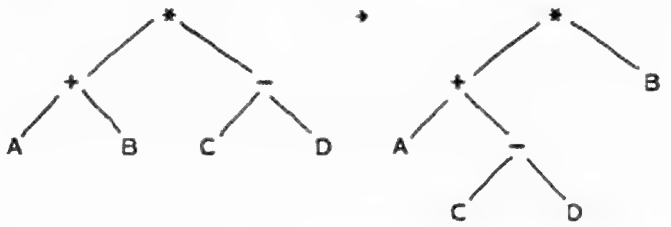


HAS ONLY ONE VIRTUAL INSTANCE, NAMELY + , WHICH IS PRECISELY THE TRANSFORMAND; IT HAS MULTIPLICITY SINCE TWO COPIES OF C ARE USED IN THE TRANSFORMAND. THE FORMULA

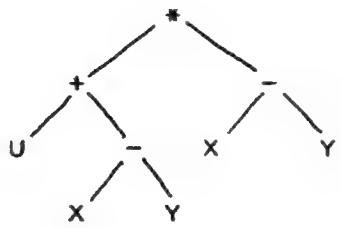


WILL BE TRANSFORMED INDEFINITELY.

THE REQUIREMENT BANNING VIRTUAL INSTANCES INSIDE THE TRANSFORMAND WHEN THEY STRICTLY CONTAIN OTHER INSTANCES WAS ALSO INTRODUCED TO COMPLETE THE PROOF. WE DO NOT KNOW EITHER IF THIS IS A NECESSARY CONDITION, BUT THERE ARE ALSO EXAMPLES THAT FAIL TO STOP AND DO NOT FULFILL THIS CONDITION. FOR INSTANCE THE PRODUCTION:



HAS NO MULTIPLICITY. BUT THE TRANSFORMAND CONSISTS OF A VIRTUAL INSTANCE WITH A COMPLEMENTARY INSTANCE ATTACHED TO IT. THEN IT DOES NOT SATISFY THE CONDITIONS OF THE THEOREM AND WE SEE THAT FOR THE INPUT FORMULA



THE PRODUCTION WILL NEVER TERMINATE.

THE THEOREM I GIVES A SUFFICIENT CONDITION FOR A SINGLE PRODUCTION TO TERMINATE. NEVERTHELESS MOST ALGORITHMS ARE NOT DESCRIBED BY A SINGLE PRODUCTION, THUS WE SHALL EXTEND OUR RESULTS TO OBTAIN A THEOREM THAT GUARANTEES THE TERMINATION OF AN ALGORITHM DEFINED BY A COLLECTION OF PRODUCTIONS.

FOR AN ALGORITHM CONSTITUTED OF SEVERAL PRODUCTIONS TO TERMINATE, IT IS NECESSARY THAT EACH PRODUCTION TERMINATES WHEN IT IS APPLIED SEPARATLY, BUT THAT IS NOT SUFFICIENT SINCE CLEARLY ONE PRODUCTION COULD UNDO THE TRANSFORMATION CARRIED OUT BY ANOTHER PRODUCTION AND THEN THESE TWO PRODUCTIONS COULD CYCLE INDEFINITELY.

LET THE SET OF PRODUCTIONS:

$$P[1] \rightarrow T[1]$$
$$\cdots$$
$$P[N] \rightarrow T[N]$$

BE DENOTED AS: (P[I]→T[I]; N).

WE OBTAIN A THEOREM THAT RESEMBLES THEOREM I. THUS WE FIRST EXTEND SOME OF OUR DEFINITIONS TO INCLUDE A COLLECTION OF PRODUCTIONS.

DEFINITION 35.- THE SET OF VIRTUAL PATTERNS OF A COLLECTION OF PRODUCTIONS (P[I]→T[I];N) IS THE UNION OF THE GENERALIZATION SETS OF ALL THE P[I]'S.

DEFINITION 36.- THE SET OF COMPLEMENTARY PATTERNS OF A COLLECTION OF PRODUCTIONS IS THE SET OF COMPLETION VALUES FOR VIRTUAL PATTERNS OF SUCH COLLECTION OF PRODUCTIONS INCLUDING GENERALIZATIONS OF THE COMPLETION VALUES.

DEFINITION 37.- A SET OF PRODUCTIONS M IS 'FINITE-APPLICABLE' TO A GIVEN FORMULA F, IF AND ONLY IF THE FORMULA F CANNOT BE TRANSFORMED INDEFINITLY, REGARDLESS OF THE ORDER OF CHOOSING PRODUCTIONS FROM M AND SELECTING BRANCHES IN F.

DEFINITION 38.- A SET OF PRODUCTIONS (P[I]→T[I];N) IS 'COLLECTIVELY COMPATIBLE' IF AND ONLY IF
1.- FOR EACH TRANSFORMAND T[I] AND EACH PATTERN P[K], IT IS THE CASE THAT

$$\neg(\ T[I]\ >>\ P[K]\ )$$

2.- IF A TRANSFORMAND T[I] HAS VIRTUAL INSTANCES THEN SUCH INSTANCES SHOULD BE RESTRICTED AS FOLLOWS:
2.1 VIRTUAL INSTANCES IN T[I] SHOULD NOT CONTAIN OTHER INSTANCES.
2.2 MULTIPLICITY CANNOT OCCUR WITHIN A VIRTUAL INSTANCE CONTAINED IN T[I].

THEOREM II. LET M BE A SET OF COLLECTIVELY COMPATIBLE PRODUCTIONS. FOR ANY FORMULA F THE SET M IS FINITE-APPLICABLE.

PROOF:

DEFINITION 39.- THE CHARACTERISTIC FUNCTION OF A FORMULA F RELATIVE TO A SET OF COLLECTIVE COMPATIBLE PRODUCTIONS IS

$$PHI(F) = TAU(F, K(F)) \qquad [39]$$

WHERE TAU AND K ARE DEFINED IN [20] AND [19]. THOSE DEFINITIONS USE THE CONCEPT OF 'INSTANCE'. IT IS UNDERSTOOD THAT IN THE PRESENT CASE, WITH A COLLECTION OF PRODUCTIONS, THE MEANING OF INSTANCES IS THAT STATED BY DEFINITIONS 35 AND 36 AND THAT A REAL INSTANCE WOULD BE ANY INSTANCE OF SOME P[I].

THE PROOF OF THIS THEOREM IS A STRAIGHTFORWARD REPLICATION OF THEOREM I'S PROOF. WE SHALL SKETCH ONLY THE MAIN POINTS OF THE PROOF.

THE CHARACTERISTIC FUNCTION PHI OF A FORMULA F IS DEFINED IN TERMS OF THE CHARACTERISTIC FUNCTIONS OF ALL STRICTLY CONTAINED INSTANCES OF F. FIRST ONE SHOWS THAT THE CHARACTERISTIC FUNCTION IS A WELL DEFINED NON NEGATIVE INTEGER VALUED FUNCTION; THIS IS DONE BY INDUCTION ON THE NUMBER OF STRICTLY CONTAINED INSTANCES IN THE FORMULA F. NEXT WE SHOW THAT EVERY TIME A PRODUCTION IS APPLIED TO ANY BRANCH OF A FORMULA THE CHARACTERISTIC FUNCTION OF THAT FORMULA DECREASES. THIS IS PROVED USING THE FOLLOWING FACTS:

1.- THE CONDITION $\neg(T[K]>>P[I])$ ASSURES THAT AFTER THE TRANSFORMATION OF A REAL INSTANCE OF P[K], THE REARRANGED BRANCH DEFINED BY T[K] WILL NOT INTRODUCE DIRECTLY A REAL INSTANCE OF SOME P[I].

2.- THE USE OF VIRTUAL INSTANCES ASSURES THAT THE TRANSFORMATION PRODUCED BY ANY PRODUCTION WILL NOT CREATE NEW INSTANCES ABOVE THE STRUCTURE BEING TRANSFORMED.

3.- THE EXPONENTIAL NATURE OF THE CHARACTERISTIC FUNCTION COMPENSATES THE MULTIPLICATION OF NEW INSTANCES. I.E. THE PHI FUNCTION GIVES MORE WEIGHT TO A SINGLE INSTANCE CONTAINING A LOT OF STRUCTURE THAN TO MANY INSTANCES CONTAINING LESS STRUCTURE.

4.- THE CONSTRAINT BANNING NESTED INSTANCES IN A TRANSFORMAND T[I] ASSURES THAT NO NEW INSTANCES WILL CONTAIN MORE STRUCTURE THAN THE REAL INSTANCE BEING ELIMINATED. BECAUSE OF THE SAME REASON MULTIPLICITY IS RESTRICTED WITHIN A VIRTUAL PATTERN IN A TRANSFORMAND.

5.- THE USE OF COMPLEMENTARY INSTANCES ASSURES THE STRICTLY DECREASING NATURE OF THE CHARACTERISTIC FUNCTION.

THEOREM II GIVES SUFFICIENT CONDITIONS FOR THE TERMINATION OF A COLLECTION OF PRODUCTIONS. UNFORTUNATELY THESE CONDITIONS ARE TOO STRONG AND MANY ALGORITHMS DO NOT FULFILL THE CONDITIONS EVEN WHEN THEY TERMINATE. FOR THIS REASON WE SHALL CONTINUE WORKING FOR A LESS RESTRICTIVE THEOREM.

DEFINITION 40.- A SET OF COLLECTIVELY COMPATIBLE PRODUCTIONS (A[I]→B[I];N) IS 'GREATER THAN' ANOTHER SET OF COLLECTIVELY COMPATIBLE PRODUCTIONS (C[J]→D[J];K), DENOTED

(A[I]→B[I]; N) >P> (C[J]→D[J]; K)

IF AND ONLY IF WHENEVER A PRODUCTION C[J]→D[J] IS APPLIED, THE CHARACTERISTIC FUNCTION OF (A[I]→B[I];N) DOES NOT INCREASE.

DEFINITION 41.- LET S[1], S[2], .., S[L] BE A LIST OF COLLECTIVELY COMPATIBLE SETS OF PRODUCTIONS. THIS LIST IS CALLED 'WELL ORDERED' IF AND ONLY IF THERE IS AN ORDER I[1], I[2], ..., I[L] OF THE INTEGERS 1, 2, .., L SUCH THAT WHENEVER AN INTEGER R APPEARS BEFORE AN INTEGER Q IN SUCH ORDERING THEN S[R] >P> S[Q].

THEOREM III. IF A SET OF PRODUCTIONS CAN BE PARTITIONED IN A WELL ORDERED LIST OF COLLECTIVELY COMPATIBLE SETS THEN FOR ANY INPUT FORMULA F THE SET OF PRODUCTION IS FINITE-APPLICABLE.

PROOF:

LET THE SET OF PRODUCTIONS BE PARTITIONED IN L DISJOINT SETS OF COLLECTIVELY COMPATIBLE PRODUCTIONS:

S[1], S[2], ... S[L]

FROM DEFINITIONS 40 AND 41 WE HAVE L CHARACTERISTIC FUNCTIONS

PHI[1], PHI[2], ... PHI[L]

SUCH THAT EVERY TIME A PRODUCTION FROM THE SET S[I] IS APPLIED THE VALUE OF PHI[I] DECREASES AND NO PHI[K] FOR K<I INCREASES ITS VALUE. SINCE THE PHI'S ARE NON NEGATIVE INTEGER VALUED FUNCTIONS THEN THE PRODUCTIONS CANNOT BE APPLIED INDEFINITELY. QED.

LET US GIVE AN EXAMPLE TO WHICH THEOREM III APPLIES.

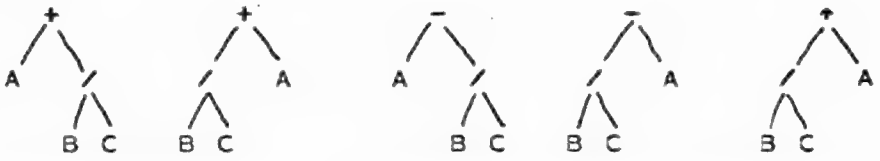
STATEMENT 4.- FOR ANY FORMULA IN CLASS Z THE ALGORITHM 'RATIO' (APPENDIX-9) TERMINATES. FURTHERMORE IT TRANSFORMS ANY FORMULA FROM CLASS TG INTO AN ALGEBRAICALLY EQUIVALENT FORMULA IN THE CLASS QG. (THESE CLASSES ARE DEFINED IN APPENDIX A-2).

PROOF:

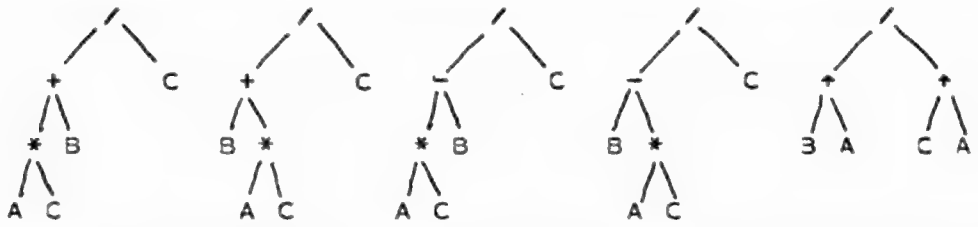
LET US CONSIDER THE PRODUCTIONS:

1.-	A + B/C	→	(A*C + B)/C
2.-	B/C + A	→	(B + A*C)/C
3.-	A - B/C	→	(A*C - B)/C
4.-	B/C - A	→	(B - A*C)/C
5.-	A*(B/C)	→	(A*B)/C
6.-	(B/C)*A	→	(B*A)/C
7.-	A/(B/C)	→	(A*C)/B
8.-	(B/C)/A	→	B/(C*A)
9.-	(B/C)↑A	→	B↑A/C↑A

NOW THE TREE STRUCTURE OF THE PATTERNS FOR PRODUCTIONS 1, 2, 3, 4 AND 9 ARE RESPECTIVELY:



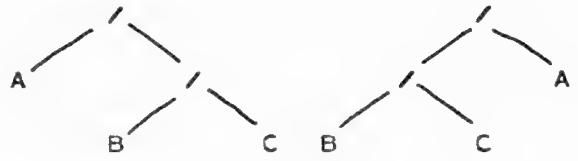
THE TRANSFORMANDS FOR THE SAME PRODUCTIONS ARE RESPECTIVELY:



THUS THE VIRTUAL PATTERNS OF THESE COLLECTION OF PRODUCTIONS ARE:

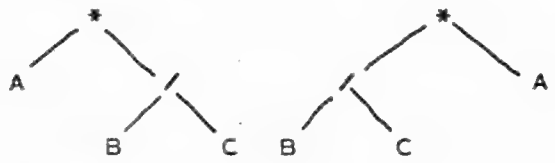
AND THE COLLECTION HAS ONLY ONE COMPLEMENTARY PATTERN: . EACH TRANSFORMAND CONTAINS SOME VIRTUAL INSTANCE; BUT NONE OF THOSE VIRTUAL INSTANCES HAS MULTIPLICITY OR CONTAINS OTHER INSTANCES. THUS PRODUCTIONS 1, 2, 3, 4 AND 9 ARE COLLECTIVELY COMPATIBLE. NOW CONSIDER PRODUCTIONS 7

AND 8, THEIR PATTERNS ARE:



THEIR TRANSFORMANDS ARE:

THIS COLLECTION HAS A VIRTUAL PATTERN: $\overset{/}{/\backslash}$ WHICH IS ALSO A COMPLEMENTARY PATTERN, THEREFORE WE CHOSE $\overset{/}{/\backslash}$ TO BE A COMPLEMENTARY PATTERN. THEN THE TRANSFORMANDS HAVE NO VIRTUAL INSTANCES AND THEREFORE PRODUCTIONS 7 AND 8 ARE COLLECTIVELY COMPATIBLE. FURTHERMORE EVERY TIME ANY ONE OF THESE TWO PRODUCTIONS IS APPLIED NO NEW INSTANCES OF $\overset{+}{/\backslash}$, $\overset{-}{/\backslash}$ OR $\overset{*}{/\backslash}$ ARE CREATED AND THE EXISTING INSTANCES OF THESE OPERATORS WILL NOT CHANGE THEIR NUMBER OF STRICTLY CONTAINED INSTANCES; THEREFORE THE CHARACTERISTIC FUNCTION OF THE SET OF PRODUCTIONS 1, 2, 3, 4 AND 9 DOES NOT INCREASE WHEN ONE OF PRODUCTIONS 7 OR 8 IS EXECUTED. FINALLY LET US CONSIDER PRODUCTIONS 5 AND 6; THEIR PATTERNS ARE:



THEIR TRANSFORMANDS ARE:

THIS LAST SET OF PRODUCTIONS HAS ONE VIRTUAL PATTERN: $\overset{*}{/\backslash}$ AND ONE

COMPLEMENTARY PATTERN: /^ ; THE TRANSFORMANDS CONTAIN A VIRTUAL INSTANCE, BUT SUCH INSTANCE HAS NO MULTIPLICITY AND DOES NOT CONTAIN OTHER INSTANCES. THUS THE SET IS COLLECTIVELY COMPATIBLE. FURTHEREMORE EVERY TIME ONE OF THESE PRODUCTIONS IS EXECUTED NO NEW INSTANCES OF +, -, * OR /^ ARE CONSTRUCTED; THEREFORE THE CHARACTERISTIC FUNCTIONS OF THE TWO PREVIOUS SETS OF COLLECTIVELY COMPATIBLE PRODUCTIONS WILL NOT INCREASE. THE ALGORITHM THUS CAN BE PARTITIONED IN THREE SETS OF WELL ORDERED COLLECTIVELY COMPATIBLE PRODUCTIONS, NAMELY:

S[1] = [PRODUCTIONS 1, 2, 3, 4 AND 9]
S[2] = [PRODUCTIONS 7 AND 8]
S[3] = [PRODUCTIONS 5 AND 6]

WITH

S[1] >P> S[2]
S[1] >P> S[3]
S[2] >P> S[3]

FROM THEOREM III THE ALGORITHM TERMINATES FOR ANY INPUT FORMULA. NOW WE SEE THAT AFTER THE ALGORITHM HAS STOPPED THERE IS NO OCUURRENCE OF OF '/'^ EXACTLY ONE LEVEL BELOW AN OCCURRENCE OF +^, -^, *^, OR /^ BECAUSE OTHERWISE ONE PRODUCTION COULD BE APPLIED WHICH CONTRADICTS THE FACT THAT THE ALGORITHM HAS TERMINATED. THEREFORE ANY BRANCH OF A FORMULA CONTAINING ONLY +^, -^, *^, /^ AND ↑^ (WITH INTEGER EXPONENT) WILL CONTAIN ONLY ONE OCCURRENCE OF '/'^ AND SUCH OCCURRENCE WILL BE THE MAIN OPERATOR OF THAT BRANCH; THUS THE BRANCH IS A MEMBER OF THE CLASS Q AND THE FORMULA WILL BE A MEMBER OF THE CLASS QG. QED.

TESTING WHETHER A GIVEN COLLECTION OF PRODUCTIONS IS COLLECTIVELY COMPATIBLE IS CLEARLY A STRAIGHTFORWARD ALGORITHM THAT COULD BE PROGRAMMED IN FORMULA ALGOL ITSELF. IT WOULD CONSIST OF THE FOLLOWING STEPS:

1.- CHECK THAT NO TRANSFORMAND T[I] CONTAINS AN INSTANCE OF SOME PATTERN P[J].

2.- COMPUTE ALL VIRTUAL PATTERNS, I.E., FOR EVERY PRODUCTION IN THE SCHEMA WE EXTRACT ALL GENERALIZATIONS OF THEIR PATTERNS. THIS IS DONE BY A SYSTEMATIC PRUNING.

3.- ONCE THE SET OF VIRTUAL PATTERNS IS CONSTRUCTED WE FORM THE SET OF ALL COMPLETION VALUES FOR SOME VIRTUAL PATTERN AND ALL GENERALIZATIONS OF SUCH COMPLETION VALUES. THIS SET IS THE SET OF COMPLEMENTARY PATTERNS.

4.- FOR EACH TRANSFORMAND T[I] CHECK THAT NO VIRTUAL INSTANCES WITHIN IT HAS EITHER MULTIPLICITY OR CONTAINS OTHER INSTANCES.

CHAPTER IV, ON LIMITING PROCESSES

WE SHALL DISCUSS SEPARATELY THE VARIOUS FEATURES OF THE PROGRAM AND AFTERWARDS DESCRIBE HOW ARE THEY ORGANIZED WITHIN THE ENTIRE PROGRAM.

WE USE SEVERAL METHODS TO COMPUTE LIMITS. THERE IS A SIMPLE STRAIGHTFORWARD ALGORITHM THAT WORKS SATISFACTORILY FOR A GREAT DEAL OF CASES THAT WE SHALL CALL TRIVIAL CASES. YET THERE ARE A CONSIDERABLE NUMBER OF SITUATIONS WHICH THIS ALGORITHM FAILS TO SOLVE; FOR SUCH SITUATIONS WE HAVE A REPERTOIRE OF METHODS THAT SEARCH FOR A SOLUTION; THIS SEARCH IS MADE SOMETIMES IN A HEURISTIC FASHION.

TRIVIAL CASES

THE ALGORITHM TO COMPUTE TRIVIAL CASES IS BASED ON A RECURSIVE TOP-DOWN ANALYSIS OF THE FORMULA TO WHICH THE LIMIT IS BEING APPLIED. THIS ALGORITHM IS MATHEMATICALLY JUSTIFIED BECAUSE OF THE WELL KNOWN THEOREMS ON LIMITS; E.G., THE LIMIT OF A SUM IS EQUAL TO THE SUM OF THE LIMITS (PROVIDED CERTAIN CONDITIONS ARE SATISFIED). ALGORITHMS OF THIS NATURE ARE EASILY WRITTEN IN FORMULA ALGOL. FOR EXAMPLE, THE THEOREM JUST MENTIONED COULD BE USED BY A FORMULA ALGOL RECURSIVE PROCEDURE, SAY PROCEDURE LIM(F), IN THE FOLLOWING WAY:

```
IF F == A: ANY + B:ANY THEN
LIM ← LIM(A) + LIM(B)  ELSE
```

OF COURSE A PROGRAM BASED ONLY ON STATEMENTS OF THIS FORM WOULD BE VERY

WEAK SINCE IT WOULD BE UNABLE TO SOLVE MANY INTERESTING PROBLEMS, NAMELY THOSE WHERE THE LIMIT OF A FUNCTION EXISTS EVEN WHEN THE LIMITS OF THE FUNCTION'S OPERANDS DO NOT EXIT, OR WHEN THE LIMITS OF THE FUNCTION'S OPERAND EXIST BUT THE FUNCTION IS UNDEFINED FOR THE VALUES THEY ATTAIN.

THE CONCEPT OF INFINITY IS INTIMATELY LINKED TO THE CONCEPT OF LIMIT. THUS IT WAS ONLY NATURAL FOR US TO ALLOW THE USE OF INFINITY AS A PERMISSIBLE VALUE IN OUR LIMITING PROCESSES. BY DEFINING LIMIT IN TERMS OF NEIGHBORHOODS AND DEFINING NEIGHBORHOOD FOR INFINITY AS WELL AS FOR FINITE VALUES WE END UP WITH A UNIFIED CONCEPT THAT MAKES NO DISTINCTION BETWEEN FINITE AND NON FINITE VALUES. WE USE THE FOLLOWING WELL KNOWN DEFINITIONS.

DEFINITION 1.- THE LIMIT OF A FUNCTION F(X) WHEN THE VARIABLE X APPROACHES THE VALUE Y IS EQUAL TO A, DENOTED

$$\lim_{X \to Y} F(X) = A$$

IF AND ONLY IF GIVEN A NEIGHBORHOOD OF A, SAY N(A), THERE IS A NEIGHBORHOOD OF Y, SAY M(Y,N), SUCH THAT

$$X \text{ IN } M(Y,N) \text{ IMPLIES } F(X) \text{ IN } N(A)$$

DEFINITION 2.- THE NEIGHBORHOOD OF RADIUS E OF A FINITE NUMBER Y IS THE SET OF NUMBERS:

$$N(Y,E) = [\, X \mid |X-Y| < E \,]$$

THE NEIGHBORHOOD OF RADIUS N OF INFINITY IS THE SET OF NUMBERS:

$$N(\text{INFINITY}, N) = [\, X \mid X > N \,]$$

THE NEIGHBORHOOD OF RADIUS N OF MINUS INFINITY IS THE SET OF NUMBERS:

$$N(-\text{INFINITY}, N) = [\, X \mid X < -N \,]$$

ONCE WE HAVE A DEFINITION FOR LIMITS THAT INCLUDES NON FINITE

VALUES WE EXTEND THE SCOPE OF THE VARIOUS ARITHMETIC OPERATORS TO INCLUDE NON FINITE OPERANDS. WE INTRODUCE SEVEN NEW SYMBOLS:

INFI+	INFI-	INFI
ZERO+	ZERO-	ZERO
	UNDEF	

THE SYMBOLS INFI+ AND INFI- REPRESENT THE REAL LINE'S TWO EXTREMES, I.E., THEY REPRESENT RESPECTIVELY A NUMBER THAT IS GREATER THAN ANY OTHER NUMBER AND A NUMBER THAT IS SMALLER (ALGEBRAICLY) THAN ANY OTHER NUMBER. THE SYMBOL INFI IS USED FOR TWO PURPOSES: ONE IS TO INDICATE THAT WE HAVE EITHER INFI+ OR INFI- BUT WE IGNORE THE SIGN; THE OTHER PURPOSE IS TO INDICATE THAT WE HAVE IN SOME SENSE BOTH VALUES. I.E., F(A)=INFI IF GIVEN K>0 THERE IS A NEIGHBORHOOD N(A) OF A SUCH THAT THERE IS A POINT X IN N(A) FOR WHICH F(X)>K AND ALSO THERE IS A POINT Y IN N(A) FOR WHICH F(Y)<-K.

THE SYMBOLS ZERO+, ZERO- AND ZERO ARE ''ISOARITHMIC'', I.E., THEY REPRESENT THE SAME NUMBER, NAMELY 0, OR IN OTHER WORDS THEY HAVE THE SAME NUMERIC VALUE; THE DISTINCTION IS MADE ON THE HISTORY OF THAT VALUE. THUS THE FIRST SYMBOL INDICATES THAT THE VALUE WAS OBTAINED BY A DECREASING SEQUENCE OF POSITIVE NUMBERS. THE SECOND SYMBOL INDICATES THAT THE VALUE WAS OBTAINED BY AN INCREASING SEQUENCE OF NEGATIVE NUMBERS. THE THIRD SYMBOL INDICATES THAT EITHER IT IS UNKNOWN HOW THE VALUE WAS OBTAINED, OR THAT THE VALUE WAS OBTAINED BY AN ALTERNATING SEQUENCE THAT DECREASES IN ABSOLUTE VALUE.

THE SYMBOL UNDEF IS USED TO INDICATE EITHER THAT THE LIMIT DOES NOT EXIST OR THAT IT IS NOT A TRIVIAL LIMIT.

TABLE I

THIS TABLE DEFINES THE MEANING OF THE ARITHMETIC OPERATORS WHEN THEY ARE USED WITH THE NEW SYMBOLS. LET US MAKE THE FOLLOWING NOTATION AGREEMENT:

A AND B STAND FOR ANY FINITE NUMBER OR ANY ONE OF THE NEW SYMBOLS.
X STANDS FOR ANY FINITE NUMBER.
I STANDS FOR ANY ONE OF INFI+, INFI- OR INFI.
I1 AND I2 STAND FOR EITHER INFI+ OR INFI-.

UNLESS OTHERWISE INDICATED THE FINITE NUMBER '0' CAN OCCUR IN ANY ONE OF ITS THREE ISOARITHMIC REPRESENTATIONS: ZERO+, ZERO- AND ZERO.

FOR PLUS:

$$A + UNDEF = UNDEF + A = UNDEF$$
$$INFI + I1 = I1 + INFI = UNDEF$$
$$I + X = X + I = I$$
$$I1 + I2 = \begin{cases} I1 & \text{IF } I1=I2 \\ UNDEF, & \text{OTHERWISE} \end{cases}$$

FOR MINUS:

$$-(INFI-) = INFI+$$
$$-(INFI+) = INFI-$$
$$-(INFI\ ) = INFI$$
$$-(ZERO-) = ZERO+$$
$$-(ZERO+) = ZERO-$$
$$-(ZERO\ ) = ZERO$$
$$-(UNDEF) = UNDEF$$
$$A - B = A+(-B)$$

FOR TIMES:

$$A * UNDEF = UNDEF * A = UNDEF$$
$$INFI * I1 = I1 * INFI = INFI$$
$$I*X = X*I = \begin{cases} I & \text{IF } X>0 \\ -I & \text{IF } X<0 \\ UNDEF & \text{IF } X=0 \end{cases}$$

FOR DIVISION:

$$1/INFI- = ZERO-$$
$$1/INFI+ = ZERO+$$
$$1/INFI = ZERO$$
$$1/ZERO = INFI$$
$$1/ZERO+ = INFI+$$
$$1/ZERO- = INFI-$$
$$1/UNDEF = UNDEF$$
$$A / B = A*1/B$$

MULTIPLICATION BY ZERO:

LET X BE ANY ONE OF ZERO+, ZERO- OR ZERO.
LET C BE ANY FINITE NUMBER.

$$C * X = X * C = \begin{cases} ZERO+ & \text{IF } (C>0 \vee C=ZERO+) \wedge (X=ZERO+) \text{ OR} \\ & \quad (C<0 \vee C=ZERO-) \wedge (X=ZERO-) \\ ZERO- & \text{IF } (C<0 \vee C=ZERO-) \wedge (X=ZERO+) \text{ OR} \\ & \quad (C>0 \vee C=ZERO+) \wedge (X=ZERO-) \\ ZERO & \text{IF } X=ZERO \text{ OR } C=ZERO \end{cases}$$

FOR POWER EXPONENTIATION:

LET X BE ANY FINITE NUMBER; LET C BE ANY VALUE OTHER THAN AN EVEN INTEGER; LET EI BE AN EVEN INTEGER.

$$X\uparrow INFI+ = \begin{cases} INFI+ & \text{IF } X>1 \\ UNDEF & \text{IF } X=1 \\ ZERO+ & \text{IF } 0<X<1 \\ ZERO+ & \text{IF } X=ZERO+ \\ & \text{OTHERWISE:} \\ UNDEF & \text{IF EXPONENTS GROWS ARBITRARILY} \\ ZERO & \begin{cases} \text{IF } X=ZERO- \\ \text{IF } X=ZERO \\ \text{IF } -1<X<0 \end{cases} \text{AND EXPONENT GROWS BY INTEGER STEPS} \\ UNDEF & \text{IF } X=-1 \\ INFI & \text{IF } X<-1 \end{cases}$$

$$INFI-\uparrow X = \begin{cases} UNDEF & \text{IF X IS NOT AN INTEGER} \\ INFI+ & \text{IF X IS AN EVEN INTEGER} \\ INFI- & \text{IF X IS AN ODD INTEGER.} \end{cases}$$

$A\uparrow UNDEF = UNDEF\uparrow A = UNDEF$
$A\uparrow INFI = INFI\uparrow C = UNDEF$
$X\uparrow INFI- = 1/(X\uparrow INFI+)$

$INFI+ \uparrow INFI+ = INFI+$
$INFI+ \uparrow INFI- = ZERO+$
$INFI \uparrow EI = INFI+$

$$INFI+ \uparrow X = \begin{cases} ZERO+ & \text{IF } X<0 \\ UNDEF & \text{IF } X=0 \\ INFI+ & \text{IF } X>0 \end{cases}$$

$$-INFI \uparrow I1 = \begin{cases} UNDEF & \text{IF THE EXPONENT CHANGES ARBITRARILY} \\ INFI & \text{IF I1=INFI+ AND IT GROWS BY INTEGER STEPS} \\ ZERO & \text{IF I1=INFI- AND IT DECREASES BY INTEGER STEPS} \end{cases}$$

FOR UNARY OPERATIONS:

ABS(UNDEF)	= UNDEF	EXP(UNDEF)	= UNDEF	LN(UNDEF)	= UNDEF
ABS(ZERO-)	= ZERO+	EXP(INFI-)	= ZERO+	LN(INFI-)	= UNDEF
ABS(ZERO+)	= ZERO+	EXP(INFI+)	= INFI+	LN(INFI+)	= INFI+
ABS(ZERO)	= ZERO+	EXP(INFI)	= UNDEF	LN(INFI)	= UNDEF
ABS(I)	= INFI+	EXP('0')	= 1	LN(ZERO)	= UNDEF
SIN(UNDEF)	= UNDEF	COS(UNDEF)	= UNDEF	LN(ZERO+)	= INFI-
SIN(I)	= UNDEF	COS(I)	= UNDEF	LN(ZERO-)	= UNDEF
SIN(ZERO+)	= ZERO+	SIN(ZERO-)	= ZERO-		

THE CORE OF THE PROGRAM THAT PROCESSES TRIVIAL LIMITS IS A FORMULA ALGOL TRANSLITERATION OF THE RULES STATED IN TABLE II. THOSE RULES ARE ACTUALLY EXTENSIONS OF THE STANDARD THEOREMS ON LIMITS. THUS IT CAN BE PROVED THAT THE VALUE YIELDED BY THE TABLE'S RULES WILL BE THE SAME THAT ONE WOULD OBTAIN IF THE DEFINITION OF LIMIT (DEF'S 1, 2) WERE USED DIRECTLY TOGETHER WITH THE NEW SYMBOLS AND THEIR RESPECTIVE MEANINGS UNDER THE VARIOUS ARITHMETIC OPERATIONS (TABLE I). THE FOLLOWING TABLE DEFINES RECURSIVELY THE RULES TO COMPUTE TRIVIAL LIMITS. LET US ASSUME WE ARE COMPUTING THE LIMIT WHEN X APPROACHES C.

TABLE II

```
LIM(   Q    ) = Q, IF Q DOES NOT CONTAIN AN OCCURRENCE OF X.
LIM( A + B ) = LIM(A) + LIM(B)        LIM( SQRT(A) ) = SQRT( LIM(A) )
LIM( A - B ) = LIM(A) - LIM(B)        LIM(  SIN(A) ) =  SIN( LIM(A) )
LIM( A * B ) = LIM(A) * LIM(B)        LIM(  COS(A) ) =  COS( LIM(A) )
LIM( A / B ) = LIM(A) / LIM(B)        LIM(  ABS(A) ) =  ABS( LIM(A) )
LIM( A ↑ B ) = LIM(A) ↑ LIM(B)        LIM(  EXP(A) ) =  EXP( LIM(A) )
LIM(   X    ) = C                     LIM(   LN(A) ) =   LN( LIM(A) )
```

THEREFORE WE COULD SAY THAT THIS PART OF THE PROGRAM IS A SPECIAL PURPOSE EVALUATOR OR INTERPRETER. A STANDARD EVALUATOR (E.G., THE BUILT-IN FORMULA ALGOL INSTRUCTION EVAL) SEQUENCES THROUGH A FORMULA ALLOWING ATOMIC FORMULAE TO BE SUBSTITUTED FOR ARBITRARY VALUES AND POSSIBLY SIMPLIFIES THE RESULT FOLLOWING THE RULES OF THE ALGEBRA OF FIELDS (E.G. X+0=0). THE PROGRAM TO COMPUTE TRIVIAL LIMITS OPERATES LIKEWISE, BUT BESIDES THE ALGEBRAIC SIMPLIFICATIONS RULES IT USES THE RULES DEFINED TO OPERATE WITH INFINITY (C.F. TABLE I).

THE LIMIT OF A FORMULA F OFTEN DEPENDS ON THE SIGN THAT THE NUMBER '0' ACQUIRES DURING THE EVALUATION OF F. THE ANALYSIS TO DETERMINE THE SIGN OF A PARTICULAR VALUE OF '0' IS IN GENERAL A VERY EXPENSIVE PROCESS (RELATIVE TO EVALUATION OF FORMULAE). FOR THIS REASON THE PROGRAM OPERATES WITH A 'SAVING EFFORT' HEURISTIC.

> HEURISTIC I. DO NOT SPEND, INITIALLY, TOO MUCH EFFORT IN LOCAL ANALYSIS. TRY TO GET A RESULT BY MEANS OF A QUICK OVERALL ANALYSIS. IN CASE OF FAILURE GO BACK AND PERFORM A MUCH MORE DETAILED JOB.

THUS THE EVALUATOR, IN A FIRST TRY, ATTEMPTS TO COMPUTE THE LIMIT WITHOUT CONCERN FOR THE SIGN OF '0'; IF THE FINAL RESULT IS 'UNDEF', 'INFI' OR 'ZERO' THEN THE EVALUATOR TRIES AGAIN BUT NOW, IT WILL INVESTIGATE THE SIGN OF '0'. FOR EXAMPLE, THIS HEURISTIC WILL PAY OFF WHEN WE COMPUTE THE LIMIT OF

$$Q(X) + (CONSTANT \neq 0)$$

AND THE LIMIT OF Q(X) HAPPENS TO BE '0'; IT WOULD NOT BE NECESSARY TO OBTAIN THE SIGN OF SUCH '0'. ON THE OTHER HAND, CONSIDER THE LIMIT (WHEN N → ZERO+) OF:

$$1/(N+2 - N) \qquad [1]$$

THE EVALUATOR WILL COMPUTE:

$$1/[\ (ZERO+)+2 - ZERO+\ ]$$

$$1/[\ ZERO+ \ - \ ZERO+\ ]$$

$$1/ZERO$$

$$INFI$$

THE HEURISTIC DID NOT WORK, IT HAS TO GO BACK AND CHECK THE SIGN OF [(ZERO+)-(ZERO+)].

LATER IN THIS CHAPTER WE SHALL DISCUSS THE VARIOUS TECHNIQUES WE USE TO DETERMINE THE SIGN OF '0'. NOW LET US EXAMINE THE METHODS UTILIZED TO COMPUTE NON-TRIVIAL LIMITS.

L' HOSPITAL'S RULE

THIS IS CLEARLY THE FIRST EXTRA FEATURE THAT ONE CONSIDERS TO INCLUDE IN A LIMITING PROCESS. IT IS PROGRAMMED AS THE PROCEDURE

LHOSPITAL(A, B)

THAT CAN BE CALLED FROM ANYWHERE IN THE PROGRAM. THIS PROCEDURE, AS WELL AS THE OTHER PROCEDURES THAT WE SHALL DISCUSS, KNOWS WHICH VARIABLE IS THE INDEPENDENT VARIABLE FOR LIMITING AND THE VALUE TO BE APPROACHED. THE PROCEDURE LHOSPITAL ASSUMES THAT THE LIMITS OF A AND B ARE BOTH ZERO OR BOTH INFINITY (OTHERWISE IT WOULD NOT HAVE BEEN CALLED). THIS PROCEDURE USES TWO HEURISTICS:

> HEURISTIC II. BEFORE EXECUTING A DIFFERENTIATION STEP, SIMPLIFY OCCURRENCES OF DIVISION OPERATORS (C.F. APPENDIX 9, MARKOV ALGORITHM 'RATIO').
>
> HEURISTIC III. IF THE RULE IS UNSUCCESSFUL IN PROCESSING A AND B THEN IT SHOULD TRY 1/B AND 1/A (SINCE A/B=[1/B]/[1/A]).

LET US TAKE AN EXAMPLE: ASSUME THAT THE PARAMETERS A AND B ARE

X AND 1/LN(X)

AND WE ARE COMPUTING THE LIMIT WHEN X→ZERO+. THE PROCEDURE COMPUTES THE DERIVATIVES:

1 AND -(1/LN(X)+2)*(1/X)

THE LIMIT OF THE SECOND EXPRESSION IS NOT DIRECTLY COMPUTABLE. THUS RATHER THAN GETTING INVOLVED WITH A POSSIBLY MORE COMPLICATED PROBLEM THAN THE ONE IT HAD TO START WITH, THE PROGRAM USES HEURISTIC III: IT CONSIDERS THE RECIPROCALS AND PERMUTES THEIR PLACES

LN(X) AND 1/X

IT COMPUTES THE DERIVATIVES:

1/X AND -1/X↑2

THE LIMIT OF BOTH EXPRESSIONS IS DEFINED, BUT THEIR QUOTIENT IS NOT. THERE IS AGAIN A SITUATION OF INFINITY OVER INFINITY; THE PROGRAM DOES NOT COMPUTE DERIVATIVES AGAIN, INSTEAD IT USES HEURISTIC II: IT SIMPLIFIES THE QUOTIENT

(1/X)/(-1/X↑2) = -X

THUS THE LIMIT IS ZERO-.

IF AFTER A SIMPLIFICATION OF DIVISION SIGNS THERE IS STILL A SITUATION OF ZERO OVER ZERO OR INFINITY OVER INFINITY THEN THE PROGRAM KEEPS ITERATING UP TO FOUR TIMES. THIS BOUND IS VERY ARBITRARY. WE CHOOSE IT WITH NO JUSTIFICATION OTHER THAN TO CUT THE TIME IN A SEARCH WHICH MIGHT SEEM FRUITLESS. NONETHELESS, IT IS IMPORTANT TO INDICATE THAT MANY CASES, WHICH THIS PROCEDURE WOULD ABANDON, ARE GOING TO BE DETECTED AND PROCESSED BY OTHER MEANS. FOR EXAMPLE, THE LIMIT WHEN X→INFI+ OF A QUOTIENT OF TWO POLYNOMIALS, ONE OF DEGREE 253 AND OTHER OF DEGREE 387, WOULD BE COMPUTED BY THE L'HOSPITAL'S RULE AFTER TWO HUNDRED AND FIFTY THREE ITERATIONS, EACH TIME COMPUTING TWO DERIVATIVES, WHICH IS CLEARLY POINTLESS. SUCH CASES ARE DETECTED BY THE ASYMPTOTIC ANALYSIS METHOD THAT WE SHALL DISCUSS NEXT.

ASYMPTOTIC ANALYSIS

THIS IS ONE OF THE FASTEST METHODS TO COMPUTE NON TRIVIAL LIMITS. AN IMPORTANT ELEMENT IN THE STUDY OF LIMITING PROCESSES IS THE RATE OF APPROACH TO THE LIMIT VALUE. PROBLEMATIC SITUATIONS ARE USUALLY ENCOUNTERED WHEN SEVERAL SUB-EXPRESSIONS APPROACH VALUES IN OPPOSITE DIRECTIONS AT DIFFERENT RATES. THE LIMIT OF THE WHOLE EXPRESSION IS OBTAINED BY ANALYSIS OF THE COMBINED EFFECT OF SUCH RATES. INDEED THE L'HOSPITAL'S RULE IS BASED ON THIS GENERAL IDEA. ANOTHER TECHNIQUE BASED ON THE SAME PRINCIPLE IS THE ASYMPTOTIC ANALYSIS.

DEFINITION 3.- TWO FORMULAE F AND G ARE 'ASYMPTOTICALLY EQUIVALENT', DENOTED

$$F \ =INFI= \ G$$

IF AND ONLY IF:

1.- $\lim_{X \to INFI+} F(X)$ EXISTS.

2.- $\lim_{X \to INFI+} G(X)$ EXISTS.

3.- $\lim_{X \to INFI+} F(X)/G(X) = 1$

COROLLARY 1.- $F \ =INFI= \ G$ IMPLIES $\lim_{X \to INFI+} F(X) = \lim_{X \to INFI+} G(X)$

COROLLARY 2.- '=INFI=' IS AN EQUIVALENCE RELATION.

ASYMPTOTIC EQUIVALENCE IS A PARTICULAR CASE OF THE 'BIG O' CALCULUS CONCEPT. WE USE COROLLARY 1 TO PRODUCE OUR NEXT HEURISTIC:

HEURISTIC IV. REPLACE COMPLEX FORMULAE BY SIMPLER ASYMPTOTICALLY EQUIVALENT FORMULAE.

THE ADJECTIVES 'COMPLEX' AND 'SIMPLER' REFER TO THE FORMULA'S TREE STRUCTURES. THUS FOR INSTANCE, HEURISTIC IV SUGGESTS REPLACING THE QUOTIENT OF TWO POLYNOMIALS BY A CONSTANT MULTIPLIED BY THE LIMITING VARIABLE RAISED TO AN INTEGER (POSITIVE OR NEGATIVE) POWER.

HEURISTC IV COULD NOT BE APPLIED TO LIMITS WHERE THE INDEPENDENT VARIABLE APPROACHES A FINITE VALUE. THE NEXT HEURISTIC COULD SOLVE SUCH A PROBLEM.

HEURISTIC V. REPLACE THE LIMIT: $\lim_{X\to A} F(X)$ BY

$\lim_{Z\to INFI+} F(A+1/Z)$ IF X→A FROM ABOVE, OR BY

$\lim_{Z\to INFI+} F(A-1/Z)$ IF X→A FROM BELOW.

REPLACE $\lim_{X\to INFI-} F(X)$ BY $\lim_{X\to INFI+} F(-X)$.

SOMETIMES THIS HEURISTIC ELIMINATES COMPLETELY THE PROBLEM. FOR EXAMPLE, CONSIDER THE LIMIT (WHEN X→1 FROM ABOVE) OF

$$(X\uparrow 2 - 1)/(X - 1)$$

WHICH IS A SIMPLE CASE FOR L'HOSPITAL'S RULE TO TAKE CARE OF. YET BY MEANS OF HEURISTIC V ONE COULD SOLVE THIS PROBLEM WITHOUT DIFFERENTIATION, ALTHOUGH WITH MUCH MORE ALGEBRAIC MANIPULATION. WE HAVE FROM HEURISTIC V THAT THE EXAMPLE IS EQUIVALENT TO

$$[(1+1/Z)\uparrow 2 - 1]/[(1+1/Z) - 1]$$

WHEN Z→INFI+. WHICH IS ALGEBRAICALLY EQUIVALENT TO

(2*Z↑3 + Z↑2)/Z↑3

THIS EXPRESSION CAN BE OBTAINED BY A FORMULA ALGOL PROGRAM THAT CALLS THE ALGEBRAIC MANIPULATION PACKAGES: 'REDUCE FRACTIONS' (APPENDIX-9); 'EXPAND PRODUCTS OVER SUMS', (APPENDIX-10); AND 'COLLECT ALIKE TERMS AND PUT THEM IN A CANNONICAL FORM', (APPENDIX-11). APPLYING HEURISTIC IV WE SEE THAT THE EXPRESSION IS ASYMPTOTICALLY EQUIVALENT TO

2

WHICH IS THE LIMIT.

IN GENERAL WHEN A FORMULA F CONTAINS ONLY THE OPERATORS +, -, *, / AND ↑ RAISED TO A NUMERIC EXPONENT, ONE DOES NOT HAVE TO USE THE EXPENSIVE PROCESS OF TRANSFORMING F INTO A QUOTIENT OF POLYNOMIALS AND THEN TAKE THE QUOTIENT OF THE LEADING TERMS. INDEED THE FOLLOWING ALGORITHM WILL OBTAIN THE SAME RESULT. LET F BE ANY FORMULA, THE ORDER

ORD(F) = [C(F), E(F)]

IS A PAIR OF VALUES DEFINED RECURSIVELY AS FOLLOWS:

TABLE III

WHEN F:	ORD(F):
HAS NO OCCURRENCE OF X	[F, 0]
IS X	[1, 1]
IS A + B	[C(A), E(A)] IF E(A)>E(B) [C(A)+C(B), E(A)] IF E(A)=E(B) AND C(A)+C(B)≠0 UNDEFINED IF E(A)=E(B) AND C(A)+C(B)=0 [C(B), E(B)] IF E(B)>E(A)
IS - A	[-C(A), E(A)]
IS A - B	ORD(A + (-B))
IS A * B	[C(A)*C(B), E(A)+E(B)]
IS A / B	[C(A)/C(B), E(A)-E(B)]
IS A ↑ B	[C(A)↑B, E(A)*B]
IS SOMETHING ELSE	UNDEFINED

WE CAN VERY EASILY CONVINCE OURSELVES THAT:

$$F(X) \ =INFI= \ C(F)*X\uparrow E(F)$$

WHENEVER C(F) AND E(F) ARE DEFINED.

CONSIDER THE LIMIT [1] THAT THE EVALUATOR FAILED TO COMPUTE. HEURISTIC V CHANGES LIMIT [1] INTO:

$$1/[\ 1/Z\uparrow 2 - 1/Z\ ] \qquad [2]$$

TABLE III'S ALGORITHM WILL PRODUCE THE ASYMPTOTIC ANALYSIS:

$$1/[\ 1/O(Z\uparrow 2) \ - \ 1/O(Z)\ ]$$

$$1/[\ O(Z\uparrow -2) \ - \ O(Z\uparrow -1)\ ]$$

$$1/[\ - O(Z\uparrow -1)\ ]$$

$$- O(Z).$$

THUS THE VALUE OF THE LIMIT IS:

$$- INFI+ \ = \ INFI-$$

WE CAN MAKE SOME COMMENTS ABOUT TABLE III'S ALGORITHM. FIRST, THE FORMULA ALGOL IMPLEMENTATION IS IMMEDIATELY OBTAINABLE FROM THE TABLE, (APPENDIX-17). SECONDLY, WE SEE THAT THE FIRST OCCURRENCE OF THE VALUE 'UNDEFINED' IS PRODUCED WHEN THE LEADING TERMS OF TWO FORMULAE CANCEL OUT EACH OTHER AND THUS THE ALGORITHM FINDS ITSELF STUCK IN A WRONG TRACK. FOR EXAMPLE, IF THE EXPRESSION

$$(1 + 1/Z)\uparrow 2 - 1$$

IS GIVEN TO THE ALGORITHM, IT WILL BEGIN THINKING THAT '1' IS THE LEADING TERM ONLY TO FIND OUT LATER, WHEN IT IS TOO LATE, THAT SUCH WAS NOT THE CASE. THE FORMULA ALGOL PROGRAM IN THIS SITUATION CALLS THE 'MULTIPLYING OUT' AND 'COLLECT ALIKE TERMS' PROCEDURES AND APPLIES THEM ONLY TO THE

SUB-EXPRESSION THAT IS CAUSING THE TROUBLE.

NOTICE THAT THE ALGORITHM IS VALID EVEN WHEN FORMULAE CONTAIN NON-NUMERIC CONSTANTS. IN THIS CASE A VERY INTERESTING PROBLEM ARISES: WHEN F IS 'A+B' THE VALUE OF ORD(F) (C.F. TABLE III) DEPENDS ON WHETHER E(A)>E(B). IF BOTH E(A) AND E(B) ARE NUMBERS, THERE IS A SIMPLE TEST TO PERFORM; OTHERWISE ONE MUST FIND AND PROVE A RELATION BETWEEN TWO FORMULAE. THIS LATTER PROBLEM WILL BE TREATED IN THE NEXT CHAPTER.

THE LAST LINE IN THE ALGORITHM ORD(F) (C.F. TABLE III) IS VERY DISAPOINTING. THIS MOTIVATES US TO INVESTIGATE THE FOLLOWING PROBLEM: GIVEN A FORMULA F, WHAT CONDITIONS WOULD LET US REPLACE SUB-EXPRESSIONS OF F BY ASYMPTOTICALLY EQUIVALENT SUB-EXPRESSIONS IN SUCH A WAY THAT THE FORMULA OBTAINED BY THE REPLACEMENT WOULD BE ASYMPTOTICALLY EQUIVALENT TO F. SUCH CONDITIONS ARE STATED IN THE 'REPLACEMENT THEOREM' THAT IS PROVED IN APPENDIX A1. THE PROCEDURE 'REPL' (APPENDIX-17) USES THE ALGORITHM OF TABLE III CORRECTED AS INDICATED ABOVE AND USES THE REPLACEMENT THEOREM.

THE PROBLEM OF KNOWING THE SIGN OF '0'

WHENEVER A SUBORDINATE PROCEDURE ARRIVES AT A SITUATION IN WHICH IT MUST KNOW THE SIGN OF A PARTICULAR VALUE '0', IT MAY USE ONE OF FIVE AVAILABLE METHODS.

THE FIRST METHOD IS BASED ON THE FACT THAT THE NUMBER 0 AND INFINITY ARE 'INVERSES'. THUS IF LIM F = '0', IT CHANGES THE PROBLEM TO FIND THE SIGN OF LIM 1/F AND UTILIZES THE ASYMPTOTIC ANALYSIS METHOD; THE

SIGN OBTAINED FOR INFINITY WITH SUCH ANALYSIS WOULD BE THE SIGN OF 0.

A SECOND METHOD APPLIES INEQUALITY ANALYSIS. SUPPOSE F(Y)=0 AND WE WANT TO FIND OUT THE SIGN OF 0. THIS METHOD SHOWS THAT ''FOR ALL X SUFFICIENTLY CLOSE TO Y'' EITHER F(X)<0 OR F(X)>0. THE IMPLEMENTATION OF THIS METHOD ALSO USES THE INVERSE TECHNIQUE BASED ON HEURISTIC V; THUS WHAT THE PROGRAM ACTUALLY SHOWS IS THAT ''FOR ALL SUFFICIENTLY LARGE Z'' EITHER F(Y+1/Z)<0 OR F(Y+1/Z)>0. THE METHODS TO FIND AND PROVE INEQUALITIES WILL BE DISCUSSED IN THE NEXT CHAPTER. THE EXAMPLE GIVEN BY LIMIT [1] AND SOLVED BY ASYMPTOTIC ANALYSIS AS LIMIT [2] CAN BE SOLVED ALSO BY THE METHOD OF INEQUALITY ANALYSIS. THE LIMIT [2] IS:

$$1/[\ 1/Z+2\ -\ 1/Z\ ]$$

WHEN Z → INFI+. WITH ALGEBRAIC MANIPULATIONS WE TRANSFORM THIS LIMIT TO:

$$Z+2/(1-Z)$$

NOW WE HAVE THAT: $Z \rightarrow$ INFI+

I.E. $Z >$ ANY NUMBER

THUS IN PARTICULAR: $Z > 1$ AND $Z > 0$

THEREFORE: $1 - Z < 0$ AND $Z+2 > 0$

THEN $Z+2/(1-Z) < 0$

SO THE SIGN IS '-'.

A THIRD METHOD IS BASED ON DIFFERENTIAL CALCULUS. WE HAVE THE FOLLOWING THEOREM:

THEOREM 1.- LET F(X) AND G(X) BE TWO FORMULAE SUCH THAT F(Y)=G(Y). THUS LIM F(X)-G(X) = 0 WHEN X→Y. LET Q=+1 INDICATE THAT X→Y FROM ABOVE AND Q=-1 INDICATE THAT X→Y FROM BELOW. IF F'(Y) ≠ G'(Y) THEN THE SIGN S OF '0' IS GIVEN BY THE TABLE:

	Q = +1	Q = -1
F'(Y)>G'(Y)	S = +1	S = -1
F'(Y)<G'(Y)	S = -1	S = +1

PROOF:

$$F'(Y) - G'(Y) = \lim_{X\to Y} \frac{F(Y) - F(X)}{Y - X} - \lim_{X\to Y} \frac{G(Y) - G(X)}{Y - X}$$

$$= \lim_{X\to Y} \frac{G(X) - F(X) - G(Y) + F(Y)}{Y - X}$$

$$= \lim_{X\to Y} \frac{F(X) - G(X)}{X - Y}$$

THUS:

$$SIGN[\ F'(Y) - G'(Y)\ ] = S/Q$$

THEN

$$S = Q * SIGN[\ F'(Y) - G'(Y)\ ]$$

THEN WE HAVE THAT:

F'(Y) > G'(Y) → SIGN[F'(Y) - G'(Y)] = +1
→ S = Q.

F'(Y) < G'(Y) → SIGN[F'(Y) - Q'(Y)] = -1
→ S = -Q.

QED.

LET US SEE HOW THIS METHOD SOLVES THE SAME EXAMPLE [1] THAT WE HAVE BEEN USING TO ILLUSTRATE DIFFERENT METHODS. WE WANT TO FIND THE SIGN OF

N↑2 - N

WHEN N → ZERO+. THUS APPLYING THEOREM 1 WE HAVE:

F(N)	=	N↑2	G(N)	=	N	Q	= +1
F'(N)	=	2*N	G'(N)	=	1		
F'(0)	=	0	G'(0)	=	1		

THUS G'(0) > F'(0) AND FROM TABLE WE HAVE S = -1.

A FOURTH METHOD TO DETERMINE THE SIGN OF ZERO IS ONE WE MIGHT CALL 'ALGEBRAIC REMOVAL OF THE DIFFICULTY'. IT CAN BE APPLIED WHEN THE VALUE '0' HAS BEEN PRODUCED BY AN ADDITION OR A SUBSTRACTION OPERATION. THIS METHOD APPLIES ONE OF THE FOLLOWING TRANSFORMATIONS

```
F + G   →   F * ( 1 + SIMPLIFY( G/F ) )
F + G   →   G * ( SIMPLIFY( F/G ) + 1 )
F - G   →   F * ( 1 - SIMPLIFY( G/F ) )
F - G   →   G * ( SIMPLIFY( F/G ) - 1 )
```

TO THE BRANCH THAT YIELDS '0'. THIS IS OF COURSE A HEURISTIC METHOD. THE TRANSFORMATIONS DO NOT GUARANTEE THE REMOVAL OF THE DIFFICULTY BUT IN MANY CASES ONE MAY SOLVE THE PROBLEM. COMING BACK TO OUR MOST EXPLOITED EXAMPLE [1] WE SEE THAT THE THIRD RULE DOES NOT WORK:

1/[N↑2 - N] = 1/[N↑2 * (1 - 1/N)]

THEN THE EVALUATOR PRODUCES:

1/[(ZERO+)↑2 * (1 - 1/ZERO+)]

1/[ZERO+ * (1 - INFI+)]

1/[ZERO+ * INFI-]

1/UNDEF

UNDEF

BUT THE FOURTH RULE DOES WORK:

1/[N↑2 - N] = 1/[N * (N - 1)]

THE EVALUATOR PRODUCES:

1/[ZERO+ * (ZERO+ - 1)]

1/[ZERO+ * (-1)]

1/ZERO-

INFI-

THE FIFTH AND LAST METHOD AVAILABLE TO DETERMINE THE SIGN OF '0' CAN BE PLACED IN THE GENERAL CLASS OF 'RECURSIVE ANALYSIS TECHNIQUES'. THERE ARE TWELVE FORMS THAT WE CAN WORK WITH AND THEREFORE THAT COULD BE ZERO:

1.	A + B = 0	7.	SIN(A) = 0
2.	A - B = 0	8.	COS(A) = 0
3.	A * B = 0	9.	EXP(A) = 0
4.	A / B = 0	10.	LN(A) = 0
5.	A ↑ B = 0	11.	ABS(A) = 0
6.	- A = 0	12.	SQRT(A) = 0

THIS METHOD DETERMINES THE SIGN OF '0' BY MEANS OF AN INDEPENDENT ANALYSIS OF EACH ONE OF THE FORMULA'S MAIN BRANCHES. THUS IT MAY CALL ITSELF RECURSIVELY. CASES 1 AND 2 ARE NOT CONSIDERED BY THIS METHOD SINCE THE VALUE'S MAGNITUDE DEPENDS ON A COMBINED RESULT OF BOTH OPERATORS (I.E. THEY CANCEL EACH OTHER). NEVERTHELESS, THE LAST TWO METHODS, WE HAVE DESCRIBED, WORK PRECISELY FOR THESE CASES. CASES 9, 11 AND 12 ARE NOT A PROBLEM SINCE THEY ALWAYS PRODUCE ZERO+. THE REMAINING CASES CONTAIN TWO KINDS OF SUBPROBLEMS: 1) DETERMINE RECURSIVELY, OR BY MEANS OF ANOTHER METHOD, THE SIGN OF '0' PRODUCED BY OPERAND A OR B. 2) FIND AND PROVE AN INEQUALITY THAT INDICATES THE SIGN OF OPERAND A OR B, WHICHEVER IS NOT '0'. THUS IN CASE 3, 4 OR 5 WE WANT TO SHOW ANY ONE OF THE INEQUALITIES.

A > 0 A < 0 B > 0 B < 0

IN CASES 7 OR 8 WE WANT TO FIND AN INTEGER N AND PROVE THE RELATIONS:

PI * N < A < PI * (N+1)

THESE TWO CASES HAVE NOT BEEN IMPLEMENTED. IN CASE 9 WE WANT TO SHOW ANY ONE OF THESE INEQUALITIES:

A > 1 A < 1

ALL THESE PROBLEMS WILL BE DISCUSSED IN THE NEXT CHAPTER. NOW LET US MOVE TO A DIFFERENT TOPIC WITHIN THE GENERAL CLASS OF METHODS AVAILABLE TO COMPUTE NON-TRIVIAL LIMITS.

THE BOUNDEDNESS TECHNIQUE

THIS TECHNIQUE IS BASED ON THE FOLLOWING THEOREM:

THEOREM. LET F(X)=M(X)*H(X); IF M(X) IS BOUNDED AND LIM H(X)=0 THEN LIM F(X) = 0.

NOTICE THAT LIM M(X) MAY BE UNDEFINED. A TYPICAL APPLICATION OF THIS THEOREM IS GIVEN BY THE LIMIT OF

SIN(X) * EXP(-X)

WHEN X→INFI+. SIN(INFI+) = UNDEF AND THERE IS NOTHING WE CAN DO ABOUT IT; THE EVALUATOR WILL NEVER WORK IN THIS CASE.

IN ORDER TO UTILIZE THIS THEOREM, WE INTRODUCED A SPECIAL ATOMIC NAME:

BOUNDED

WHICH IS A FORM VARIABLE THAT ALWAYS HAS ITS OWN NAME AS VALUE. THE EVALUATOR DOES NOT UNDERSTAND THIS WORD, I.E., IT DOES NOT SIMPLIFY OCCURRENCES OF THE WORD 'BOUNDED', NEVERTHELESS IT GENERATES OCCURRENCES OF THAT WORD IN THE FOLLOWING CASES.

```
SIN( INFI+ ) = BOUNDED      COS( INFI+ ) = BOUNDED
SIN( INFI- ) = BOUNDED      COS( INFI- ) = BOUNDED
SIN( INFI  ) = BOUNDED      COS( INFI  ) = BOUNDED
```

THESE RULES ARE THUS USED INSTEAD OF THOSE DEFINED IN TABLE I. IN ORDER TO USE THE ABOVE THEOREM WE HAVE A RECURSIVE PROCEDURE (APPENDIX-18) THAT TESTS FOR BOUNDEDNESS OF A FORMULA. SUCH A PROCEDURE IS BASED ON PRODUCTIONS OF THE FORM

```
BOUNDED + BOUNDED       →    BOUNDED
BOUNDED - BOUNDED       →    BOUNDED
BOUNDED * BOUNDED       →    BOUNDED
BOUNDED / BOUNDED[¬0]   →    BOUNDED
        - BOUNDED       →    BOUNDED
    SIN(    ANY    )    →    BOUNDED
    COS(    ANY    )    →    BOUNDED
    ABS( BOUNDED )      →    BOUNDED
    EXP( BOUNDED )      →    BOUNDED
    LN( BOUNDED[>0])    →    BOUNDED
```

THE ABOVE THEOREM IS A SPECIAL CASE OF THE NEXT THEOREM THAT ALSO MIGHT BE USEFUL, ALTHOUGH WE HAVE NOT IMPLEMENTED IT.

THEOREM. LET F(X), G(X) AND H(X) BE THREE FORMULAE SUCH THAT:
1. F(X) ¬< G(X) ¬< H(X).
2. LIM F(X) = LIM H(X) = Q.
THEN LIM G(X) = Q.

THIS THEOREM WOULD ALLOW US TO AVOID VERY COMPLICATED CASES OF LIMIT APPROACHING, PROVIDED ONE CAN FIND TWO FUNCTIONS THAT BOUND THE TEST CASE AND SUCH THAT THEIR LIMITS ARE THE SAME.

SEQUENCES

DETERMINING THE LIMIT OF A SEQUENCE IS A RATHER COMMON PROCESS. NEVERTHELESS, FROM THE MECHANIZATION VIEWPOINT, THIS CASE INTRODUCES INTERESTING PROBLEMS. IN THE FIRST PLACE, TECHNIQUES LIKE L'HOSPITAL'S RULE THAT ARE BASED ON DIFFERENTIATION CAN NO LONGER BE USED. ALTHOUGH SOME OTHER TECHNIQUES (E.G. 'ASYMPTOTIC ANALYSIS', 'ALGEBRAIC REMOVAL OF THE DIFFICULTY', 'INEQUALITY ANALYSIS') STILL CAN BE USED. ON THE OTHER HAND, SOME LIMITS ARE ONLY DEFINED IN THIS CASE. FOR INSTANCE

$$A \uparrow B$$

IN REAL VARIABLE ANALYSIS IS NOT DEFINED WHEN A<0 AND B IS ARBITRARY, BUT IS WELL DEFINED IF B IS RESTRICTED TO HAVE INTEGER VALUES (C.F. TABLE I). THUS WHILE COMPUTING THE LIMIT OF A SEQUENCE A NEW KIND OF SUB-PROBLEM MAY ARRISE: PROVE THAT A FORMULA IS INTEGER VALUED. WE HAVE PROGRAMMED A RECURSIVE PROCEDURE CALLED 'ENTERO' (APPENDIX-15) THAT PROVES 'INTEGERNESS' BY MEANS OF RULES OF THE FORM:

```
INTEGER + INTEGER          → INTEGER
INTEGER - INTEGER          → INTEGER
INTEGER * INTEGER          → INTEGER
INTEGER ↑ [INTEGER ¬< 0]   → INTEGER
        - INTEGER          → INTEGER
     ABS(INTEGER)          → INTEGER
```

THE BASIS OF THE RECURSION IS GIVEN BY TWO KINDS OF CONDITIONS: THE FORMULA'S VALUE IS AN INTEGER NUMBER OR IT IS A FORMULA THAT BY DEFINITION IS INTEGER VALUED. FOR THE LATTER CASE, INFORMATION FROM OUTSIDE IS SUPPLIED.

SINCE SEQUENCES ARE FUNCTIONS WITH INTEGER VALUED DOMAINS, SOME SUB-PROBLEMS MIGHT BE SOLVED VERY EASILY BY MEANS OF INDUCTION. WE SHALL SEE ONE SUCH CASE (PROVE F(N)>0) IN THE NEXT CHAPTER.

WITH THIS TOPIC WE CONCLUDE OUR REVIEW OF TECHNIQUES TO COMPUTE WHAT WE HAVE CALLED NON TRIVIAL LIMITS. LET US SEE NEXT THE FORMULA ALGOL PROGRAMS THAT MECHANIZE SUCH TECHNIQUES.

ORGANIZATION OF THE PROGRAM

THE PROGRAM WE HAVE CONSTUCTED TO COMPUTE LIMITS IS THE FORMULA ALGOL PROCEDURE

LIMIT(F, X, T, DIREC)

WHERE F IS A FORMULA VARIABLE WHOSE VALUE MAY BE ANY ARITHMETIC FORMULA; X IS A FORMULA VARIABLE WHOSE VALUE MUST BE AN ATOMIC VARIABLE; T IS A FORMULA VARIABLE WHOSE VALUE MAY BE ANY ARITHMETIC FORMULA; AND DIREC IS AN INTEGER VARIABLE WHOSE DOMAIN IS [-1, 0, 1]. THE VALUE OF THE ABOVE PROCEDURE CALL WOULD BE THE FORMULA (SPECIAL CASE: NUMBER) OBTAINED BY COMPUTING:

$$\lim_{X \to T} F(X) \qquad \text{IF DIREC} = 0$$

$$\lim_{X \to T^{+}} F(X) \qquad \text{IF DIREC} = 1$$

$$\lim_{X \to T^{-}} F(X) \qquad \text{IF DIREC} = -1$$

THE PROCEDURE LIMIT, BESIDES THE ACTUAL PARAMETERS, GETS INFORMATION ABOUT THE PROBLEM TO BE SOLVED BY MEANS OF THREE GLOBAL LISTS:

'ENTFUN' , 'GREAT' AND 'BOUND'

WHICH CONTAIN AUXILIARY HYPOTHESES OR INFORMATION ABOUT THE ENVIRONMENT IN WHICH THE PROBLEM IS GOING TO BE SOLVED. THE LIST 'ENTFUN' CONTAINS FORMULA VARIABLES OR FORMULAE THAT ARE BY DEFINITION INTEGERS. FOR EXAMPLE, IF THE VARIABLE N IS IN THE LIST 'ENTFUN' AND WE ASK FOR THE LIMIT WHEN N→INFI+ THEN FROM TABLE I WE SEE THAT

(-0.5)↑N

IS WELL DEFINED, NAMELY 'ZERO'. THE LIST 'GREAT' CONTAINS INEQUALITIES THAT BY DEFINITION ARE TRUE. THUS WE CAN HELP THE PROGRAM TO SOLVE THE PROBLEM OF FINDING THE SIGN OF A VALUE BY STORING IN 'GREAT' AUXILIARY INFORMATION OR HYPOTHESES, SUCH AS A>0. THE LIST 'BOUND' CONTAINS FORMULAE OR VARIABLES THAT BY DEFINITION ARE BOUNDED. AN IMPORTANT CHARACTERISTIC OF ALL THREE LISTS IS THAT THEY CAN CONTAIN, NOT ONLY PARTICULAR FORMULAE, BUT ALSO A PATTERN DEFINING A CLASS OF FORMULAE. FOR EXAMPLE, THE LIST 'BOUND' ALWAYS CONTAINS THE PATTERNS:

SIN(ANY) AND COS(ANY)

THE THREE 'ISOARITHMIC' REPRESENTATIONS OF '0' AND THE THREE NON-FINITE VALUES, THAT WE HAVE INTRODUCED, BEHAVE IN A RATHER SIMILAR FASHION (C.F. TABLE I). BECAUSE OF THIS REASON MUCH OF THE CASE ANALYSIS NEEDED TO MANIPULATE THOSE SYMBOLS CAN BE REDUCED, PROVIDED ALL SIX SYMBOLS ARE ENCODED IN AN APPROPRIATE MANNER. A VERY CONVENIENT WAY TO DO SO, IS BY GIVING A SINGLE NAME TO ALL OF THEM AND ATTACHING A 'DESCRIPTION LIST' TO EACH OCCURRENCE OF SUCH A NAME WITHIN A FORMULA.

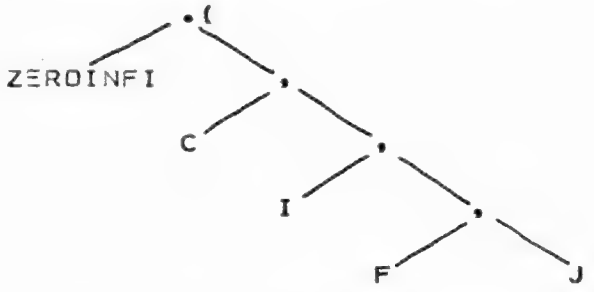
UNFORTUNATELY, FORMULA ALGOL DOES NOT HAVE, AS YET, LOCAL DESCRIPTION LISTS. A DESCRIPTION LIST CAN BE GIVEN TO A DECLARED VARIBLE, SAY Y, BUT WE CANNOT ATTACH A DESCRIPTION LIST TO EACH OCCURRENCE OF THE NAME Y WITHIN A FORMULA. NEVERTHELESS, BY MEANS OF A DOT-PROCEDURE WE CAN ACHIEVE AN THE EFFECT OF A DESCRIPTION LIST. WE DECLARE THE PROCEDURE:

ZEROINFI(C, I, F, J)

WHICH IS A SELF-GENERATING PROCEDURE, I.E., ITS BODY CONSISTS OF THE SINGLE STATEMENT:

ZEROINFI ← ZEROINFI.(C, I, F, J)

THEREFORE EVERY TIME THIS PROCEDURE IS USED IN THE CONSTRUCTION OF A FORMULA, A TREE STRUCTURE IS ACTUALLY PRODUCED WHICH COULD BE INTERPRETED AS A DESCRIPTION LIST ATTACHED TO THE SYMBOL 'ZEROINFI':



THE PARAMETER J DISTINGUISHES BETWEEN '0' AND INFINITY:

J=0 WHEN ZEROINFI ACTS AS '0'
J=1 WHEN ZEROINFI ACTS AS INFINITY.

THE PARAMETER C GIVES THE SIGN (EXCEPT AS INDICATED BELOW). THIS PARAMETER MUST BE A NON-ZERO NUMBER OR AN ARBITRARY FORMULA NOT CONTAINING THE LIMITING VARIABLE X; IN THE LATTER CASE THE SIGN OF ZEROINFI MAY DEPEND ON DIFFERENT ASSIGNED VALUES TO VARIABLES OF C. THE PARAMETER I IS A NON-ZERO NUMBER WHICH DESCRIBES THE 'SPEED' USED TO OBTAIN THE VALUE OF '0' OR INFINITY. THE PARAMETER F DESCRIBES THE NATURE

OF ZEROINFI AS FOLLOWS:

1) F=0, ZEROINFI IS ASYMPTOTICALLY EQUIVALENT TO C*Z↑I.

2) F=1, THE BEHAVIOR OF ZEROINFI IS UNKNOWN, BUT ITS SIGN IS THAT OF C.

3) F=2, BOTH THE BEHAVIOR AND THE SIGN OF ZEROINFI ARE UNKNOWN.

4) F = FORMULA CONTAINING THE VARIABLE Z. ZEROINFI IS ASYMPTOTICALLY EQUIVALENT TO C*F(Z) WHERE F(Z) IS POSITIVE, THUS THE SIGN OF ZEROINFI IS STILL THAT OF C.

NOW, THE PROGRAM CONSISTS BASICALLY OF THREE PROCEDURES:

1. MAIN PROGRAM (LIMIT PROCEDURE)
2. RESTRICTED LIMIT ROUTINE (RLIM PROCEDURE)
3. AUXILIARY LIMIT ROUTINE (AUXLIM)

(C.F. APPENDIX-14). BESIDES THERE IS AVAILABLE A COLLECTION OF PROCEDURES THAT CAN BE CALLED FROM ANYWHERE, THEY ARE:

1. THE EVALUATOR.
2. TEST SIGN OF A FORMULA.
3. TEST SIGN OF ZERO.
4. PROVE INEQUALITY.
5. PROVE BOUNDNESS.
6. TEST FOR INTEGER VALUED FUNCTION.
7. SYMPLIFICATION PACKAGES.

THE MAIN PROGRAM.

THIS PROGRAM IS A VERY SIMPLE DRIVER WHOSE FUNCTION IS CHECKING WHETHER A RESTRICTED LIMIT IS DESIRED OR NOT, I.E., IT CHECKS IF THE VARIABLE IS RESTRICTED TO APPROACH FROM BELOW OR FROM ABOVE. IF THAT IS THE CASE, IT TRANSFERS DIRECTLY TO THE 'RLIM' ROUTINE. OTHERWISE, IT CALLS SUCH A ROUTINE TWICE, ONCE WITH THE VARIABLE APPROACHING FROM ABOVE

AND A SECOND TIME WITH THE VARIABLE APPROACHING FROM BELOW. AFTERWARDS IT CHECKS THAT BOTH LIMITS ARE THE SAME.

THE RESTRICTED LIMIT ROUTINE.

THIS PROCEDURE IS STILL A DRIVER. ALTHOUGH MORE SOPHISTICATHED THAN THE MAIN PROGRAM. THE DUTIES OF THIS ROUTINE ARE TWOFOLD: TO PREPARE THE VALUE OF 'T' WHICH THE LIMITING VARIABLE X IS APPROACHING; AND TO ADMINISTRATE HEURISTIC I. THUS IF THE VALUE OF T IS A FORMULA, THIS ROUTINE WILL SIMPLIFY IT AS MUCH AS POSSIBLE; OR IF THE VALUE OF T IS IS ANY ONE OF THE SYMBOLS ZERO+, ZERO-, INFI+ OR INFI- THIS ROUTINE WILL ENCODE THEM RESPECTIVELY AS FOLLOWS:

```
ZEROINFI( 1,-1, 0, 0 )
ZEROINFI(-1,-1, 0, 0 )
ZEROINFI( 1, 1, 0, 1 )
ZEROINFI(-1, 1, 0, 1 )
```

ONCE THE RLIM ROUTINE HAS PREPARED THE VALUE OF T, IT CALLS THE AUXILIARY LIMIT ROUTINE WHICH DOES MOST OF THE WORK. THE RLIM ROUTINE REMAINS AS A MONITOR ON THE AUXLIM ROUTINE BY MEANS OF THE 'CARE' MODES. THERE IS A BOOLEAN PARAMETER CALLED 'CARE' THAT IS SET, BY PROCEDURE RLIM, INITIALLY TO FALSE. WHEN CARE IS FALSE ALL PROCEDURES THAT ACCOMPLISH LOCAL TASKS WILL NOT EXPEND TOO MUCH EFFORT TO DETERMINE THE SIGN OF VALUES. THE IDEA IS, TO AVOID GETTING INVOLVED WITH A VERY DIFFICULT LOCAL PROBLEM THAT POSSIBLY IS NOT RELEVANT TO THE FINAL SOLUTION. FOR EXAMPLE, THERE IS NO POINT TO DETERMINE THE SIGN OF A '0' THAT WILL BE ELIMINATED BY AN ADDITION TO A NON-ZERO VALUE; OR THERE IS

NO POINT TO DETERMINE THE SIGN OF AN INFINITY THAT IS GOING TO BE SQUARED. BUT SINCE THERE IS NOT ENOUGH INFORMATION, AT THE PLACE WHERE A VALUE IS COMPUTED, TO DECIDE WHETHER THE SIGN OF SUCH A VALUE WILL MATTER IN THE FINAL RESULT, WHAT HEURISTIC I PROPOSES, IS TO LOOK VERY CASUALLY AT THE PROBLEM AS A WHOLE AND HOPE FOR A SOLUTION. NOW IF THE FINAL VALUE IS EITHER ZERO, INFI OR UNDEF THEN THE RLIM ROUTINE SETS THE VALUE OF CARE TO TRUE AND TRIES AGAIN.

THE PROPAGATION OF A 'CAREFUL' MODE IS NOT INSTANTANEOUS. WHEN CARE IS TRUE EVERY LOCAL TASK MUST FIND ITS BEST ANSWER. BUT EACH LOCAL TASK ALSO USES HEURISTIC I. THUS THEY WILL EACH TRY TO ACCOMPLISH THEIR JOB INITIALLY WITHOUT CARE; AND ONLY IF THEY ARE UNSUCCESSFUL THEY WILL TRY AGAIN WITH CARE.

THE AUXILIARY LIMIT ROUTINE.

THIS ROUTINE IS THE CORE OF THE COMPUTATION FOR LIMITS. IT RECIVES INFORMATION AND GENERAL POLICY GUIDE LINES FROM THE RLIM ROUTINE. NEVERTHELESS THE AUXLIM ROUTINE IS THE ONE WHICH MAKES MOST DECISIONS AND THEREFORE KNOWS ABOUT ALL AVAILABLE METHODS AND THEIR REQUIREMENTS. THE OPERATION OF THE AUXLIM ROUTINE CAN BE STATED IN A FEW WORDS: IT TRIES TO COMPUTE THE LIMIT USING TABLE II'S ALGORITHM, HOPING FOR A TRIVIAL LIMIT. IF IT IS UNSUCCESSFUL, FROM THE DIAGNOSTICS OF THE FAILURE IT PICKS UP ALTERNATIVE METHODS THAT COULD EITHER ELIMINATE THE DIFFICULTY FOR TABLE II'S ALGORITHM OR COMPUTES THE LIMIT DIRECTLY BY OTHER MEANS. THE MAIN STEPS OF AUXLIM'S ALGORITHM ARE:

1.- IF F IS A CONSTANT OR IF IT IS X, GIVE THE LIMIT DIRECTLY AND EXIT; OTHERWISE

2.- EXTRACT F'S OPERATOR AND OPERANDS.

3.- COMPUTE RECURSIVELY THE LIMITS OF THE OPERANDS.

4.- IF SOME OPERAND'S LIMIT CANNOT BE COMPUTED, GO TO STEP 9; OTHERWISE

5.- CALL THE EVALUATOR TO CARRY OUT THE OPERATION DEFINED BY F'S OPERATOR AND THE LIMITS OF F'S OPERANDS.

6.- IF THE EVALUATOR IS UNABLE TO PRODUCE AN ANSWER GOTO STEP 8; OTHERWISE

7.- EXIT WITH THE VALUE GIVEN BY THE EVALUATOR.

8.- TAKE DIFFERENT ACTIONS DEPENDING ON THE 'FAILURE REPORT' FILED BY THE EVALUATOR:

8.1 F IS A+B; LIM A+B IS UNDEFINED ONLY WHEN LIM A AND LIM B ARE BOTH NON-FINITE AND DISTINCT. THE FORMULA ALGOL NOTATION FOR THIS ACTION IS SELF EXPLANATORY.

```
D ← LHOSPITAL( B, A );
IF ¬( D == -1) THEN
AUXLIM ← LIM(A) * (1 + D) ELSE
AUXLIM ← LHOSPITAL(  A, 1/(1 + B/A);
```

THE THIRD LINE OF THIS PIECE OF PROGRAM IS NOT WHAT ACTUALLY WAS CODED SINCE THE OPERATIONS OF '*' AND '+' ARE ACCOMPLISHED THROUGH THE EVALUATOR. SUCH A LINE DESCRIBES, NEVERTHELESS, MUCH BETTER THE IDEA BEHIND THE ALGORITHM. IN THE PROGRAM (APPENDIX-14) THE FOLLOWING WAS ACTUALLY CODED:

```
AUXLIM( 3, LIMA, LI( 1, 1, D, CARE), CARE )
```

8.2 F IS A*B; THIS CASE IS UNDEFINED WHEN THE LIMIT OF ONE OPERAND IS ZERO AND THE LIMIT OF THE OTHER IS NON-FINITE. THE ROUTINE AUXLIM EXECUTES ONE STATEMENT:

AUXLIM ← LHOSPITAL(A, 1/B)

8.3 F IS A/B. THIS CASE IS UNDEFINED ONLY WHEN THE LIMIT OF THE OPERANDS ARE BOTH '0' OR BOTH NON-FINITE. THE AUXLIM ROUTINE EXECUTES THE STATEMENT:

AUXLIM ← LHOSPITAL(A, B)

8.4 F IS A↑B AND THE LIMIT IS OF THE FORM: 1↑(INFI+), (ZERO+)↑(ZERO+), (INFI+)↑(ZERO+). THE ROUTINE EXECUTES THE STATEMENT:

AUXLIM ← LI(12, LHOSPITAL(1/B, LN(A)), ANY, CARE)

WHICH IS A CALL TO THE EVALUATOR TO COMPUTE:

EXP(LHOSPITAL(1/B, LN(A)))

8.5 F IS A↑B AND LIMIT(A) IS NEGATIVE. THE ROUTINE AUXLIM CALLS THE PROCEDURE 'ENTERO' WHICH TESTS WHETHER B IS AN INTEGER NUMBER OR AN INTEGER VALUED FUNCTION (APPENDIX-15); IF THAT IS THE CASE THE AUXLIM ROUTINE PRODUCES:

[(-1)↑LIMIT(B)] * [(- LIMIT(A))↑LIMIT(B)]

THE FIRST FACTOR IS SIMPLIFIED TO -1, 1 OR 'BOUNDED' IF LIMIT(B) IS AN ODD INTEGER, AN EVEN INTEGER, OR INFI+ RESPECTIVELY. IF LIMIT(B) IS A FORMULA, SUCH A FACTOR REMAINS AS IT IS.

8.6 F IS A↑B AND LIMIT(B) IS 'INFI'. THUS THE PROBLEM IS TO INVESTIGATE THE SIGN OF SUCH 'INFI'; THE AUXLIM ROUTINE CALLS THE TESTSIGN ROUTINE.

8.7 F IS LN(A) OR SQRT(A) AND LIMIT(A) HAS AN UNKNOWN SIGN. THE AUXLIM ROUTINE CALLS THE 'TESTSIGN' ROUTINE.

8.8 THE LIMIT OF F IS 'ZERO' AND WE ARE IN THE 'CAREFUL' MODE. THE AUXLIM ROUTINE CALLS THE 'ZEROSIGN' ROUTINE.

8.9 THE VALUE IS UNDEFINED AND THERE IS NOTHING WE CAN DO ABOUT IT. E.G. LN(<NEGATIVE NUMBER>). THE ROUTINE EXITS AND SETS THE BOOLEAN VARIABLE 'HOPE' TO FALSE; THIS WAY, ITS SUPERVISOR WILL NOT WASTE ITS TIME IN A FRUITLESS ATTEMPT TO REMOVE THE PROBLEM.

9. IF MODE IS 'CARELESS', EXIT AND LET THE SUPERVISOR KNOW, AS SOON AS POSSIBLE, ABOUT THE PROBLEM. OTHERWISE CONTINUE TO THE NEXT STEP.

10. IF PROBLEM IS OF THE FORM:

'UNDEF' * '0'

TRY TO PROVE THAT THE LEFT OPERAND IS BOUNDED. OTHERWISE CONTINUE TO THE NEXT STEP.

11. CONSTRUCT AN ASYMPTOTICALLY EQUIVALENT PROBLEM. THIS STEP USES HEURISTIC V, HEURISTIC IV, TABLE III'S CORRECTED ALGORITHM, AND THE REPLACEMENT THEOREM. IF THE PROGRAM FAILS AT THIS POINT IT CONCEDES AND QUITS WITH THE VALUE 'UNDEF'.

THE EVALUATOR

THE EVALUATOR IS MAXLIM'S MAIN SUBORDINATE. ITS MAIN RESPONSABILITY IS TO MAP AN OPERATOR AND SOME VALUE(S) INTO ANOTHER

VALUE. THEREFORE IT KNOWS, AMONG OTHER THINGS, ABOUT TABLE I'S RULES. THE EVALUATOR ALSO USES, THE BEST IT CAN, THE INFORMATION STORED IN THE 'ZEROINFI' DUAL REPRESENTATION OF '0' AND INFINITY DESCRIBED AT THE BEGINING OF THIS SECTION. FOR EXAMPLE, FROM TABLE I WE HAVE THAT:

'0' * INFINITY = UNDEF

WHEN THE EVALUATOR FINDS SUCH A SITUATION IT DOES NOT QUIT AND LET ITS SUPERVISOR WORRY ABOUT IT. INSTEAD, IT TRIES TO SOLVE THE PROBLEM BY ITSELF. LET US SUPPOSE THAT THE ABOVE '0' AND INFINITY HAVE THE REPRESENTATION:

ZEROINFI(-8, -3, 0, 0) * ZEROINFI(A↑2, 2.5, 0, 1)

I.E. THE '0' IS A ZERO- ASYMPTOTICALLY EQUIVALENT TO -8*Z↑-3 AND THE 'INFINITY' IS AN INFI+ ASYMPTOTICALLY EQUIVALENT TO A↑2*Z↑2.5, BUT THE EVALUATOR KNOWS ALSO TABLE III'S ALGORITHM. THUS IT WILL GIVE THE ANSWER:

ZEROINFI(-8*A↑2, -0.5, 0, 0)

I.E., A ZERO- WHICH APPROACHES '0' AS -8*A↑2/Z↑(0.5). ON THE OTHER HAND, IF THE EVALUATOR COULD NOT FIGURE OUT THE VALUE OF '0'*INFINITY, IT WOULD QUIT; BUT ITS OUTPUT WOULD NOT BE A LACONIC 'UNDEF'. INSTEAD IT WILL PASS ALONG ALSO INFORMATION DESCRIBING THE CIRCUMSTANCES OF THE FAILURE AND THUS IT MIGHT GIVE A GOOD HINT TO ITS SUPERVISOR. IN THIS CASE THE HINT WOULD BE: ''TRY L|HOSPITAL'S RULE PRECEDED BY THE TRANSFORMATION A*B = A/(1/B)''.

THE EVALUATOR RECOGNIZES THE SEVERAL FORMS THAT A VALUE CAN ATTAIN. THIS WAY, IT IS ABLE TO CHOOSE THE APPROPRIATE RULE TO OPERATE WITH. THEREFORE THE EVALUATOR USES TABLE I'S RULES IF A 'ZEROINFI' VALUE IS FOUNDED. IF NUMERIC VALUES ARE TO BE COMBINED, THE EVALUATOR CHECKS

THE VALIDITY OF THEIR RANGE (E.G. THE OPERATOR 'SQRT' MUST HAVE A NON NEGATIVE ARGUMENT) AND AFTERWARDS TRANSFERS TO FORMULA ALGOL'S EVAL ROUTINE WHICH MAKES THE COMPUTATION. IF THE OPERANDS ARE BOTH FORMULAE IT ALSO TRANSFERS TO EVAL. FINALLY THE EVALUATOR SOMETIMES MAKES ASSUMPTIONS ABOUT THE NATURE OF A FORMULA WHOSE CHARACTERISTICS ARE UNKNOWN. FOR EXAMPLE, IF A FORMULA 'A' IS MULTIPLIED BY '0' IT SIMPLIFIES IT TO '0' BUT ALSO PRINTS:

'' ASSUME A ≠ INFINITY ''

THE ORGANIZATION OF THE PROGRAM ALLOWED US TO TEST SEPARATELY ITS VARIOUS PARTS AND EXPERIMENT WITH DIFFERENT PROGRAMMING TECHNIQUES. ALSO, AS WE SHALL SEE NEXT, THIS ORAGANIZATION PERMITTED US TO SEGMENT THE PROGRAM SO IT COULD FIT IN THE MACHINE'S MAIN MEMORY.

PERFORMANCE OF THE PROGRAM

THE PROGRAM HAS BEEN TESTED WITH MANY EXAMPLES. SOME OF THEM ARE LISTED IN THE APPENDIX-24. THESE EXAMPLES WERE TAKEN FROM AN ADVANCED CALCULUS BOOK [33]. WE ALSO TRIED PROBLEMS THAT WERE GIVEN IN A 'HOUR-EXAM' TO FRESHMEN AT CARNEGIE TECH. IN THIS LATTER GROUP THERE WERE PROBLEMS THAT WERE GIVEN TO STUDENTS WHEN THEY HAD NOT YET LEARNED L'HOSPITAL'S RULE, SO THAT IN THE PROBLEM'S SOLUTION THEY WERE NOT SUPPOSED TO USE SUCH A TECHNIQUE. OUR PROGRAM SUCCESSFULLY FOUND THOSE LIMITS USING ONLY THE ASYMPTOTIC ANALYSIS TECHNIQUE TOGETHER WITH ALGEBRAIC TRANSFORMATIONS. FOR EXAMPLE, THE PROGRAM USING ASYMPTOTIC

ANALYSIS FOUND THE LIMIT OF

$$[\ (X+H)/(1\ +\ H\ +\ X)\ \ -\ \ X/(X\ +\ 1)\ ]/H$$

WHEN H→ZERO+, (APPENDIX-25). BUT IN THIS SAME EXAMPLE, WHEN THE PROGRAM WAS NOT FORCED TO USE ASYMPTOTIC ANALYSIS IT FAILED TO FIND THE CORRECT LIMIT. IT DID NOT REALIZE THAT THE QUANTITY IN SQUARE BRACKETS BECOMES '0' WHEN H IS REPLACED BY ZERO+. THE PROGRAM PRINTED THE ABSURD STATEMENT:

$$\text{'' ASSUME (} X/(X+1) - X/(X+1) \text{) } \neq 0 \text{ ''}$$

AND IT GAVE THE ANSWER

$$\text{SIGN(} X/(1+X) - X/(1+X) \text{)*INFIP}$$

THIS MISTAKE AND SOME OTHER DIFFICULTIES ARE PRODUCED BY LIMITATIONS IN THE SPACE FOR COMPILING CODE THAT FORBID US TO PUT TOGETHER ALL THE METHODS AND ROUTINES IN A SINGLE RUN. THUS WE CANNOT ACCOMPLISH A SOPHISTICATED SIGN ANALYSIS AT THE SAME TIME WE ARE COMPTUTING LIMITS. THESE LIMITATIONS WILL BE DISCUSSED IN THE LAST CHAPTER.

CHAPTER V. ON INEQUALITIES AND ALGEBRAIC 'SIMPLIFICATION'

IN THIS CHAPTER WE DESCRIBE SOME ALGEBRAIC MANIPULATION TASKS THAT ARE NECESSARY NOT ONLY TO AID THE EVALUATION OF LIMITS, BUT THAT ARE ALSO NEEDED TO SOLVE OTHER PROBLEMS IN MATHEMATICAL ANALYSIS. THUS FOR EXAMPLE, WE ARRIVED AT THE PROBLEM OF DETECTING AND PROVING INEQUALITIES BECAUSE OF ITS RELATION WITH THE PROBLEM OF DETERMINING THE SIGN OF A FORMULA. YET THE MANIPULATION OF INEQUALITIES IS ALSO REQUIRED BY DIVERSE PROBLEMS. E.G. THE EPSILON-DELTA TYPE OF PROOFS, THE PROOFS OF MONOTONICITY AND THE VARIOUS SUB-PROBLEMS CONNECTED WITH THE PROOF OF CONVERGENCE IN A SERIES. THE PROBLEM OF MECHANIZING THE MANIPULATION OF INEQUALITIES, AS FAR AS WE KNOW, HAS NOT BEEN TREATED IN THE LITERATURE.

ALMOST ALL TASKS IN MATHEMATICAL DATA PROCESSING NEED THE HELP OF SOME SORT OF ALGEBRAIC 'SIMPLIFICATION'. THE COMPUTATION OF LIMITS IS CERTAINLY NO EXCEPTION. WE INDICATED EARLIER IN THIS DISSERTATION THAT FORMULA ALGOL HAS ALMOST NO BUILT-IN SIMPLIFICATION ROUTINES. THUS WE HAVE PROGRAMMED A COLLECTION OF PROCEDURES AND MARKOV ALGORITHMS THAT PROVIDE REASONABLY GOOD SIMPLIFICATION CAPABILITIES.

THESE SIMPLIFICATION ROUTINES WILL BE DESCRIBED LATER IN THIS CHAPTER. NOW LET US PROCEEDE TO DESCRIBE OUR FINDINGS ON THE PROBLEM OF MANIPULATION OF INEQUALITIES.

THE PROGRAM WE USE TO FIND AND PROVE INEQUALITIES IS THE FORMULA ALGOL PROCEDURE:

FINDRELAT(F, G, Y);

WHERE F AND G ARE FORMULAE AND Y IS A LIST THAT CONTAINS THE ATOMIC VARIABLES OF F AND G TOGETHER WITH THEIR HEURISTIC BOUNDS OF VARIATION. THE PURPOSE OF THIS PROCEDURE IS TO FIND A RELATION OPERATOR ⟨R⟩ AND TO PROVE F ⟨R⟩ G FOR ALL POSSIBLE VALUES OF THE VARIABLES IN THE FORMULAE F AND G, I.E., IT PROVES THAT F⟨R⟩G IS A TAUTOLOGY. THE RELATION OPERATORS ARE:

⟨R⟩ ::= > | ≥ | =

THE PROCEDURE 'FINRELAT' ALSO CAN PROVE BY MEANS OF NUMERICAL COUNTEREXAMPLES THAT THERE IS NO RELATION OPERATOR ⟨R⟩ FOR WHICH F ⟨R⟩ G IS IDENTICALLY TRUE.

WE HAVE MENTIONED IN CHAPTER IV THAT THERE IS A LIST CALLED 'GREAT' THAT INITIALLY CONTAINS RELATIONS THAT ARE TRUE BY DEFINITION. DURING THE EXECUTION OF THE PROGRAM ALL THE RELATIONS THAT ARE PROVED BY THE VARIOUS AUXILIARY PROCEDURES WILL BE SAVED IN THE SAME LIST 'GREAT'. THUS THIS LIST CONTAINS AT ANY MOMENT ALL THE PROGRAM'S KNOWLEDGE OF EXISTING INEQUALITIES. THIS IS A VERY PRIMITIVE AND INEFFICIENT 'LEARNING DEVICE'. A MORE SOPHISTICATED 'LEARNING-FORGETING' MECHANISM COULD BE PROGRAMMED WITH EASE IN FORMULA ALGOL, BUT OUR INTENTIONS ARE TO EXPLORE THE ALGEBRAIC MANIPULATION SIDE OF THE PROBLEM RATHER THAN ITS ARTIFICIAL INTELLIGENCE SIDE. OUR PROGRAM NEVER FORGETS WHAT IT HAS

LEARNED AND IT LEARNS EVERY INEQUALITY THAT IT HAPPENS TO PROVE WITH MORE THAN ONE STEP .

THE LIST 'GREAT' CONTAINS NOT ONLY PARTICULAR INEQUALITIES, BUT ALSO CONTAINS THEOREMS' SCHEMATA THAT ARE DESCRIBED BY FORMULA ALGOL PATTERNS. FOR EXAMPLE, THE TRIANGLE INEQUALITY OCCURS IN THE LIST 'GREAT' AS THE PATTERN

ABS(OF(BA) + OF(BB)) < ABS(OF(BA)) + ABS(OF(BB))

THE CONSTRUCTIONS 'OF(BA)' AND 'OF(BB)' MATCH ANY FORMULA THE FIRST TIME THEY ARE APPLIED AND THE MATCHED VALUE IS SAVED. ALL SUBSEQUENT CALLS OF SUCH CONSTRUCTIONS WILL MATCH ONLY FORMULAE THAT ARE EQUAL TO THE FIRST FORMULA FOUND. THIS PATTERN BEHAVIOR CAN BE PROGRAMMED IN FORMULA ALGOL IN THE FOLLOWING WAY. FIRST, WE DEFINE TWO PROCEDURES:

```
BOOLEAN PROCEDURE BA(F); VALUE F; FORM F;
BA ← F == EA;
BOOLEAN PROCEDURE BB(F); VALUE F; FORM F;
BB ← F == EB;
```

THE VARIABLES 'EA' AND 'EB' ARE GLOBAL FORMULA VARIABLES. BEFORE EACH THEOREM SCHEMA IS TRIED OUT THE VARIABLES 'EA' AND 'EB' ARE INITIALIZED TO 'EA:ANY' AND 'EB:ANY' RESPECTIVELY. THUS THE FIRST TIME THE PROCEDURE BA IS EXECUTED IT WILL PRODUCE THE VALUE TRUE (I.E. A SUCCESSFUL MATCH) SINCE THE VALUE OF EA IS THE PATTERN 'ANY'. BUT EA HAD ALSO AN EXTRACTOR TO ITSELF, SO THE NEXT TIME THE PROCEDURE BA IS EXECUTED IT WOULD PRODUCE A TRUE VALUE ONLY IF F IS EQUAL TO THE FIRST VALUE MATCHED BY PROCEDURE BA.

BY MEANS OF THE INFORMATION STORED IN THE LIST 'GREAT' WE CAN NOT ONLY HELP THE PROGRAM TO SOLVE PROBLEMS, BUT ALSO WE CAN INDUCE SPECIAL MODES OF PROOF. FOR INSTANCE, WHEN THE LIST 'GREAT' CONTAINS THE PATTERN

N > ANY

WE ARE INDICATING IN A SIMPLE WAY THAT WE WANT TO PROVE AN INEQUALITY UNDER THE ASSUMPTION OF SUFFICIENTLY LARGE N. SIMILARLY WE HAVE, FOR EXAMPLE, THE PROBLEM OF PROVING A RELATION BETWEEN F(X) AND G(X) FOR SUFFICIENTLY SMALL X. WE PRECEDE THE CALL OF THE PROCEDURE 'FINREL AT' WITH THE THREE STATEMENTS:

```
G ← SUBS(X)G(1/Z);
F ← SUBS(X)F(1/Z);
INSERT  Z>ANY BEFORE 1ST OF GREAT;
```

THE FIRST TWO STATEMENTS MAKE A SUBSTITUTION OF X FOR 1/Z IN THE FORMULAE F AND G. (C.F. HEURISTIC V IN PREVIOUS CHAPTER). THE THIRD STATEMENT INFORMS THE PROGRAM THAT WE WANT TO PROVE THE RELATION FOR ARBITRARY LARGE Z, I.E., FOR ARBITRARY SMALL X.

WHEN THE RELATION TO BE FOUND IS BETWEEN FORMULAE THAT CONTAIN AN INTEGER VARIABLE, SAY N, AN INDUCTION PROOF CAN BE ARRANGED IN A NATURAL WAY. BY MEANS OF NUMERICAL EVALUATION AND A NUMERICAL TEST WE COULD PROVE, LET US SAY:

F(1) > G(1)

NEXT, WE INSERT THE RELATION F(N)>G(N) IN THE LIST 'GREAT', THUS GIVING THE INDUCTION HYPOTHESIS. FINALLY, WE SEEK TO PROVE PROVE

F(N+1) > G(N+1)

A PROOF FOR MONOTONICITY OF F(X) CAN ALSO BE ARRANGED IN A NATURAL WAY. WE INSERT A>B IN THE LIST 'GREAT' AND WE ASK THE PROCEDURE TO PROVE F(A)>F(B).

A VERY POWERFUL TECHNIQUE TO 'SOFTEN' DIFFICULTIES IN A PROOF IS CASE ANALYSIS. THE PROGRAM ITSELF CAN MAKE SEVERAL ASSUMPTIONS THAT COVER ALL POSSIBILITIES AND PROVE THE INEQUALITY SEVERAL TIMES, EACH TIME USING ONE ASSUMPTION. A TYPICAL EXAMPLE WOULD BE WHEN A RELATION CONTAINING THE VARIABLE V CANNOT BE PROVED DIRECTLY. ONE CAN VERIFY THE RELATION FOR V=0 AND PROVE IT FOR TWO CASES; IN EACH CASE INSERTING IN THE LIST 'GREAT' ONE OF THE RELATIONS: V>0, V<0.

THE LIST PROCESSING CAPABILITIES OF FORMULA ALGOL PROVIDE AN EXCELLENT COLLECTION OF TOOLS TO UPDATE THE PROGRAM'S ENVIRONMENT. THESE CAPABILITIES ARE NOT DESCRIBED IN THIS DISSERTATION (C.F. INTRODUCTION TO CHAPTER II), BUT LET US MENTION JUST A FEW OF THE STATEMENTS THAT CAN BE USED. NOTICE THAT THEY ARE SELF-EXPLANATORY.

```
INSERT  A>B  BEFORE THE J TH OF GREAT;
DELETE LAST OF GREAT;
DELETE ALL OF(HASX) OF GREAT;
```

A AND B ARE FORMULA VARIABLES; J IS AN INTEGER VARIABLE; GREAT IS A LIST VARIABLE; HASX IS A BOOLEAN PROCEDURE THAT IS TRUE WHEN ITS INPUT IS A FORMULA THAT HAS AT LEAST ONE OCCURRENCE OF X. ALL OTHER WORDS IN THE ABOVE STATEMENTS ARE FORMULA ALGOL RESERVED IDENTIFIERS. FINALLY, AS WE HAVE ALREADY SEEN IN CHAPTER II, THE BOOLEAN PRIMARY

```
F > G  ==  OF(GREAT)
```

IS TRUE WHENEVER F > G IS A RELATION CONTAINED IN THE LIST GREAT.

SO FAR WE HAVE DESCRIBED VARIOUS PERFORMANCES OF THE PROGRAM THAT CAN BE ARRANGED BY GIVING THE APPROPRIATE AUXILIARY INFORMATION. NOW LET US DESCRIBE HOW THE PROGRAM FINDS AND PROVES INEQUALITIES. FOR THIS PURPOSE WE HAVE STUDIED FOUR METHODS OF PROOF:

1.- PROOF BY ANALYSIS.

2.- PROOF BY A LOGICAL CHAIN.

3.- PROOF BY TRANSITIVITY.

4.- PROOF BY SYNTHESIS.

THE PROCEDURE 'FINDRELAT' IS THE MAIN DRIVER OF THE VARIOUS METHODS OF PROOF (APPENDIX-20); ITS MAIN STEPS ARE:

1.- TRY TO FIND AND PROVE A RELATION BETWEEN F AND G USING THE ANALYSIS METHOD. WHEN SUCCESSFUL IT EXITS; OTHERWISE IT GOES TO THE NEXT STEP.

2.- APPLY THE PROCEDURE 'GUESS'. THIS IS THE HEURISTIC HINT STEP; THIS PROCEDURE GENERATES RANDOM VALUES BETWEEN THE BOUNDS INDICATED IN THE LIST Y FOR EACH ONE OF THE VARIABLES OF F AND G. WITH THESE NUMERIC VALUES THE PROCEDURE PRODUCES SEVERAL PARTICULAR CASES THAT ALLOW IT EITHER TO PROVE BY MEANS OF COUNTEREXAMPLES THAT NO RELATION HOLDS BETWEEN F AND G OR TO MAKE A CONJECTURE TO BE PROVED.

3.- IF A CONJECTURE IS MADE THE PROCEDURE 'PROVECONJ' WILL BE CALLED WHICH WILL USE THE VARIOUS METHODS OF PROOF.

PROOF BY ANALYSIS

THIS METHOD PERFORMS A RECURSIVE TOP-DOWN ANALYSIS OF THE FORMULAE F AND G. IT BREAKS THE MAIN PROBLEM INTO SUB-PROBLEMS WHOSE SOLUTION MIGHT IMPLY A RELATION BETWEEN F AND G. THUS WHEN THIS METHOD IS

SUCCESSFUL IT WILL GENERATE A 'BACKWARDS PROOF'.

LIKE ALL RECURSIVE ALGORITHMS, THIS METHOD HAS TWO MAIN STEPS THAT WE CALL BOTTOM CONDITIONS AND RECURSIVE RULES.

I) BOTTOM CONDITIONS.

THESE STOP THE RECURSIVE SEARCH BY SOLVING A SUB-PROBLEM DIRECTLY USING OUTSIDE KNOWLEDGE. A RESULT IS PRODUCED AND THE ALGORITHM EXITS USUALLY TO POP UP ONE LEVEL OF RECURSION. SUCH BOTTOM CONDITIONS ARE:

1.- A RELATION INVOLVING F AND G EXISTS IN THE LIST 'GREAT' AND THEREFORE SUCH A RELATION HAS BEEN GIVEN EITHER AS AN AUXILIARY HYPOTHESIS OR IT IS A RESULT ALREADY PROVED BY THE PROGRAM.

2.- F AND G SATISFY THE CONDITIONS OF A THEOREM SCHEMA.

3.- F AND G ARE NUMBERS. THERE IS A WELL DEFINED RELATION BETWEEN THEM.

4.- F AND G HAVE DISTINCT SIGNS. THUS FOR EXAMPLE, F>0 AND G<0 IMPLIES F>G.

5.- F AND G ARE IDENTICAL IN STRUCTURE. THUS F=G.

6.- F AND G ARE ATOMIC ENTITIES AND NO RELATION IS KNOWN BETWEEN THEM. THE ALGORITHM EXITS WITH FAILURE.

II) RECURSION RULES.

WHEN AT LEAST ONE OF THE FORMULAE F OR G HAS STRUCTURE AND ALL THE CONDITIONS 1, 2, 4 AND 5 ABOVE HAVE FAILED THEN THE ALGORITHM CONTINUES AS FOLLOWS.

1.- THE MAIN OPERANDS AND OPERATORS OF F AND G ARE EXTRACTED.

2.- THE ALGORITHM AT THIS POINT IS DRIVEN BY A SWITCHING TABLE THAT CONTAINS FOR VARIOUS OPERATOR PAIRS, A PROCEDURE THAT DESCRIBES A CONTINUATION UNDER THOSE CIRCUMSTANCES.

3.- THE SWITCHING TABLE MAY BE EMPTY FOR SOME ENTRIES, BUT IN

GENERAL CONTAINS A PROCEDURE THAT RECURSIVELY WILL TRY TO PROVE A SET OF RELATIONS BETWEEN THE BRANCHES OF F AND G THAT WOULD IMPLY A RELATION BETWEEN F AND G. FOR EXAMPLE, IF F IS A+B AND G IS C+D THEN A>C AND B>D IMPLIES F>G. THE ALGORITHMS FOR SOME ENTRIES OF THE TABLE MIGHT DO SOME COMBINATORIAL SEARCH. FOR INSTANCE, IN THE ABOVE EXAMPLE A>D AND B>C ALSO WOULD IMPLY F>G.

NOTICE THAT THIS ALGORITHM CAN BE USED NOT ONLY TO PROVE A REALATION BETWEEN F AND G, BUT ALSO CAN BE USED TO FIND A RELATION BETWEEN F AND G.

THE IMPLICATIONS THAT THE ANALYSIS METHOD USES ARE MOST OF THE RULES DESCRIBED IN BECKENBACH AND BELLMAN'S [3] INTRODUCTORY BOOK. TO DESCRIBE SUCH IMPLICATIONS MORE CONCISELY LET US MAKE THE FOLLOWING NOTATION AGREEMENT.

<R1> ::= > | ≥ | =

<R2> ::= > | ≥ | =

LET <R3> BE DEFINED FROM <R2> AND <R1> AS FOLLOWS:

<R1> \ <R2>	>	≥	=
>	>	>	>
≥	>	≥	≥
=	>	≥	=

IMPLICATIONS FOR '+' :

A <R1> C |
> IMPLIES A + B <R3> C + D
B <R2> D |

WHERE A, B, C OR D CAN BE ZERO. I.E. THEY MAY NOT APPEAR IN THE RIGHT

SIDE OF THE IMPLICATION. THE ALGORITHM CONSIDERS PERMUTATIONS OF THE TERMS.

IMPLICATIONS FOR '-' :

IT USES THE TRANSFORMATION: A - B → A + (-B)
AND THE IMPLICATION: A > B → -B > -A.

IMPLICATIONS FOR '*' :

A <R1> C > 0 |
> IMPLIES A * B <R3> C * D
B <R2> D > 0 |

WHERE A, B, C OR D CAN BE '1'. I.E. THEY MAY NOT APPEAR IN THE RIGHT SIDE OF THE IMPLICATION. THE ALGORITHM BESIDES CONSIDERING PERMUTATION OF FACTORS, ALSO CONSIDERS THE POSSIBILITY OF A FACTOR 1=(-1)*(-1) DISTRIBUTED IN EACH FACTOR. THUS WHEN <R1> AND <R2> ARE BOTH '>' WE HAVE EIGHT CASES THAT MIGHT PRODUCE A*B > C*D; THEY ARE:

A > B > 0 C > D > 0	A > D > 0 C > B > 0
-A > B > 0 -C > D > 0	-A > D > 0 -C > B > 0
A > -B > 0 C > -D > 0	A > -D > 0 C > -B > 0
-A > -B > 0 -C > -D > 0	-A > -D > 0 -C > -D > 0

IMPLICATIONS FOR '/' :

WE USE TWO ALTERNATIVES. THE FIRST IS THE TRANSFORMATION

A/B → A*(1/B)

FOLLOWED BY THE USE OF THE IMPLICATIONS FOR '*'. THE SECOND ALTERNATIVE IS THE IMPLICATION

```
                A  <R1>  C  >  0   |
                                   >   IMPLIES   A / B  <R3>  C / D
                D  <R2>  B  >  0   |
```

WHERE B AND D CAN BE '1', I.E., THEY MAY NOT APPEAR IN THE IMPLICATION'S RIGHT SIDE.

IMPLICATIONS FOR '↑' :

WHEN A>0 AND B>0 THEN WE USE

```
        B*LN(A)  <R1>  D*LN(C)   IMPLIES   A↑B  <R1>  C↑D
```

WHERE B OR D CAN BE '1'. I.E. THEY MAY NOT APPEAR IN THE RIGHT SIDE OF THE IMPLICATION.

THIS METHOD ALSO USES SOME GENERAL FACTS. FOR INSTANCE IT KNOWS:

```
        -1  <  SIN( ANY )  <  1

        -1  <  COS( ANY )  <  1
```

THE METHOD IS AWARE OF THE FACT THAT WHEN F(X) IS MONOTONIC INCREASING

```
            A  <R1>  B   IMPLIES   F(A)  <R1>  F(B)
```

FURTHERMORE, THE METHOD KNOWS THAT LN(X), EXP(X) AND SQRT(X) ARE MONOTONIC INCREASING. FINALLY THIS METHOD USES THE IMPLICATIONS

```
            A  <R1>  B   IMPLIES   EXP(A)  >  B
            A  <R1>  B   IMPLIES    A  >  LN(B)
```

AND THE TRANSFORMATIONS:

```
            A  >  0   IMPLIES   ABS(A) = A
            A  <  0   IMPLIES   ABS(A) =-A
```

PROOF BY LOGICAL CHAIN

THIS METHOD PROVES THE RELATION R[0] BY MEANS OF A CHAIN OF RELATIONS

$$R[0] \leftrightarrow R[1] \leftrightarrow R[2] \leftrightarrow \ldots \leftrightarrow R[N]$$

WHERE EACH RELATION R[I] IS CONNECTED TO ITS NEIGHBORS BY AN 'IF AND ONLY IF' TYPE OF ARGUMENT. R[N] IS PROVED BY ANALYSIS.

THE REASON FOR THIS METHOD IS CLEAR: WHEN A RELATION R[I] CANNOT BE PROVED BY ANALYSIS WE CREATE INSTEAD A RELATION R[I+1] WHICH IS LOGICALLY EQUIVALENT AND HOPEFULLY WOULD BE EASIER TO PROVE. THIS METHOD IS BASICALLY ITERATIVE, CONTRASTING WITH THE ANALYSIS METHOD WHICH IS BASICALLY RECURSIVE.

THE SUCCESS OF THIS METHOD DEPENDS ON ITS CLEVERNESS IN GENERATING EQUIVALENT RELATIONS. THE GENERATION OF SUCH RELATIONS COULD BE VIEWED AS A SIMPLIFICATION PROBLEM SINCE THE GOAL IS TRANSFORMING A RELATION INTO ANOTHER WHICH IS SIMPLER, AT LEAST AS FAR AS THE PROOF BY ANALYSIS IS CONCERNED. AS IN ALL SIMPLIFICATION PROBLEMS, IT IS SO THAT A GIVEN SET OF TRANSFORMATIONS MIGHT BE VERY USEFUL IN SOME CASES, WHILE IN OTHERS THEY MIGHT PRODUCE UNDESIRABLE RESULTS. FOR THIS REASON, THIS METHOD CANNOT HAVE A FIXED SET OF A PRIORI RULES. THE PROGRAM SHOULD BE ABLE TO CHOOSE AND APPLY DIFFERENT TRANSFORMATION RULES TO GENERATE EQUIVALENT RELATIONS.

FOR THE PURPOSES OF THE ALGORITHM USED TO IMPLEMENT THIS METHOD OF PROOF WE HAVE CLASSIFIED IN TWO GROUPS THE TRANSFORMATIONS THAT

GENERATE NEW PROBLEMS.

I. TRANSFORMATIONS THAT PRODUCE LOGICALLY EQUIVALENT RELATIONS WHOSE CORRESPONDING PARTS ARE NOT ALGEBRAICALLY EQUIVALENT. FOR EXAMPLE

$$0 < A < SQRT(B) \quad \leftrightarrow \quad A\uparrow 2 < B$$

NEITHER A AND A↑2 NOR SQRT(B) AND B ARE ALGEBRAICALLY EQUIVALENT.

II. TRANSFORMATIONS THAT PRODUCE LOGICALLY EQUIVALENT RELATIONS PRECISELY BECAUSE THEIR CORRESPONDING ELEMENTS ARE ALGEBRAICALLY EQUIVALENT. FOR EXAMPLE

$$A\uparrow 2 - 2*A*B + B\uparrow 2 > C \quad \leftrightarrow (B-A)\uparrow 2 > C$$

WE HAVE PROGRAMMED A COLLECTION OF MARKOV ALGORITHMS THAT GENERATE EQUIVALENT RELATIONS FOLLOWING DIFFERENT CRITERIA. A PARTICULAR ALGORITHM FOR A PROOF BY LOGICAL CHAIN IS CONSTRUCTED BY CHOOSING TWO SETS OF PRODUCTIONS (MARKOV ALGORITHMS) FROM THIS COLLECTION. THE FIRST SET OF PRODUCTIONS WOULD GENERATE EQUIVALENT PROBLEMS OF TYPE I ABOVE. WE SHAL CALL IT AN ''IFF GENERATOR'' THE SECOND SET OF PRODUCTIONS WOULD GENERATE EQUIVALENT PROBLEMS OF TYPE II ABOVE. WE SHALL CALL IT A ''SIMPLIFICATOR GENERATOR''. THE MEANING OF SIMPLIFICATION IS NOT FIXED BUT IS DEFINED BY THE ALGORITHM CHOSEN TO ACT AS THE SIMPLIFICATOR GENERATOR. THUS THE PROGRAM MAY USE AT DIFFERENT EXECUTIONS COMPLETELY OPPOSITE CRITERIA FOR SIMPLIFICATION AS WOULD THE ALGORTITHM 'EXPAND' (APPENDIX-10) AND THE PROCEDURE 'FACTOR' (APPENDIX-12). THE METHOD OF PROVING BY LOGICAL CHAIN WAS IMPLEMENTED (APPENDIX-22) FOLLOWING THESE STEPS:

1.- CHOOSE TWO GENERATORS AND PLUG THEM IN AS 'IFF' AND 'SIMPLIFICATOR'

2.- APPLY 'IFF'. WHEN SUCCESSFUL IN CREATING A NEW PROBLEM CONTINUE TO NEXT STEP. OTHERWISE GO TO STEP 5.

3.- TRY TO PROVE THE INEQUALITY GENERATED USING THE ANALYSIS METHOD.

4.- IF SUCCESS, EXIT. OTHERWISE GOTO STEP 2.

5.- APPLY THE SIMPLIFICATOR. WHEN SUCCESSFUL IN CREATING A NEW PROBLEM GO TO STEP 3. OTHERWISE EXIT WITH FAILURE.

NOTICE THAT THE ALGORITHM ASSUMES THAT A TRANSFORMATION OF TYPE I (IFF) IS MORE DESIRABLE THAN ONE OF TYPE II. THIS IS SO BECAUSE THE FACT THAT CORRESPONDING ELEMENTS ARE NOT ALGEBRAICALLY EQUIVALENT SUGGESTS THAT SOME INTRINSIC DIFFICULTY HAS BEEN REMOVED.

THERE IS A VARIATION FOR THE CHAIN METHOD: INSTEAD OF GENERATING A CHAIN OF 'EQUIVALENT PROBLEMS' GENERATE A CHAIN OF REVERSE IMPLICATIONS OR 'SUFFICIENT CONDITIONS'

$$R[0] \leftarrow R[1] \leftarrow R[2] \leftarrow \ldots \leftarrow R[N]$$

THUS R[N] IS NOT EQUIVALENT TO R[0], BUT PROVING R[N] WOULD IMPLY THE VALIDITY OF R[0].

'SUFFICIENT CHAIN' ALGORITHMS ARE ORGANIZED IN THE SAME WAY AS 'EQUIVALENT CHAIN' ALGORITHMS. BUT THERE IS A NEW STEP THAT SHOULD PRECEED STEP 3 ABOVE. THIS NEW STEP SHOULD CHECK HEURISTICALLY THE VALIDITY OF A NEW PROBLEM GENERATED BY A SUFFICIENT STEP. FOR EXAMPLE, SUPPOSE THE MARKOV ALGORITHM 'SUFF' CONTAINS A PRODUCTION

$$A + POS > B \rightarrow A > B$$

WHICH INDICATES THAT IF THE PROBLEM IS TO SHOW THAT THE FORMULA 'A' PLUS

SOME POSITIVE QUANTITY IS GREATER THAN THE FORMULA 'B'. IT WOULD BE SUFFICIENT TO SHOW THAT A IS GREATER THAN B. BUT THERE IS NO GUARANTEE THAT A>B IS A TRUE RELATION. BEFORE GOING AHEAD AND TRYING TO PROVE A>B THE ALGORITHM WILL DO SOME NUMERICAL CHECKING AND VERIFY WHETHER THAT IS A REASONABLE CONJECTURE. WHEN THE NUMERICAL CHECKING INDICATES THAT THE CONJECTURE IS FALSE, THE PRODUCTION THAT PRODUCED SUCH A CONJECTURE WILL BE REMOVED FROM THE MARKOV ALGORITHM, BUT ONLY FOR THAT EXECUTION. THE PROCESS CONTINUES BY LOOKING FOR A MATCH WITH ANOTHER PRODUCTION.

APPENDIX-13 CONTAINS TWO MARKOV ALGORITHMS THAT PRODUCE EQUIVALENT PROBLEMS OF TYPE I. USING THESE ALGORITHMS THE PROGRAM PROVED THE CONJECTURE

$$2/(1/B + 1/A) < SQRT(B*A)$$

SUBJECT TO THE ASSUMPTIONS A>0 AND B>0. THE ACTUAL COMPUTER OUTPUT IS GIVEN IN APPENDIX-23.

PROOF BY TRANSITIVITY

THIS METHOD IS BASED ON THE SOLUTION OF A COMPLETELY NEW KIND OF PROBLEM. LET US ASSUME THAT OUR GOAL IS TO PROVE F>G, THE PROBLEM IS TO FIND A FORMULA H SUCH THAT

$$F > H > G$$

THE FINDING OF A FORMULA H CLEARLY REQUIRES A HEURISTIC SEARCH. THERE ARE MANY WAYS TO DIRECT THE SEARCH; WE CHOSE A WAY WHICH CONVERGES VERY RAPIDLY, I.E., IT CHECKS A VERY SMALL NUMBER OF FORMULAE THAT ARE CREATED BY TWO HEURISTIC RULES.

1.- CHECK WHAT WE KNOW ALREADY ABOUT F AND G.

2.- WHEN WE REMOVE SOME SUB-EXPRESSION OF A FORMULA A IT IS LIKELY THAT THE MUTILATED FORMULA WILL BE EITHER ALWAYS GREATER OR ALWAYS SMALLER THAN A.

AN ALGORITHM FOR THIS METHOD OF PROOF HAS BEEN CODED IN FORMULA ALGOL (APPENDIX-19, PROCEDURE 'TRANSPROOF'). THIS PROCEDURE USES A LIST CALLED 'PH' WHICH CONTAINS FORMULAE THAT WILL BE CONSIDERED AS POSSIBLE CANDIDATES FOR THE INTERMEDIATE FORMULA H. THE MAIN STEPS OF THE ALGORITHM ARE:

1. SEARCH IN THE LIST OF KNOWN FACTS 'GREAT' FOR RELATIONS THAT MIGHT CONTAIN F OR G IN THE APPROPRIATE SIDE OF THE INEQUALITY, I.E., TEST EACH MEMBER OF THE LIST GREAT AGAINST THE FORMULA ALGOL PATTERNS:

F > H:ANY AND H:ANY > G

ALL SUCH FORMULAE H WILL BE STORED IN THE LIST PH. (HEURISTIC RULE 1).

2.- PUT THE MAIN SUB-EXPRESSIONS OF F AND G AT THE END OF LIST PH. (HEURISTIC RULE 2).

3.- TAKE PH'S FIRST FORMULA AS A POSSIBLE H. IF THE LIST PH IS EMPTY THEN EXIT WITH FAILURE. OTHERWISE CONTINUE TO THE NEXT STEP.

4.- CHECK WITH NUMERICAL VALUES FOR A POSSIBLE SET OF RELATIONS BETWEEN F, H AND G. (PROCEDURE 'HEURISTICTEST', APPPENDIX-19).

5.- TRANSFER TO ONE OF THE NEXT FIVE PLACES, DEPENDING ON THE PREVIOUS RESULT.

5.1 THERE ARE NO UNIQUE RELATIONS BETWEEN F, H AND G. DISCARD THAT H AND GO TO STEP 3.

5.2 F<H AND H<G. ALARM|. THIS SHOWS THAT THE ORIGINAL CONJECTURE F>G IS FALSE. EXIT WITH ALARM SET.

5.3 F>H AND H>F. GOOD|. GO AHEAD AND TRY TO FORMALLY PROVE THE TWO NEW CONJECTURES USING THE COMBINED POWER OF ALL METHODS OF PROOF. IF ANY ONE OF THOSE CONJECTURES CANNOT BE PROVED, DISCARD THAT H AND GO TO STEP 3.

5.4 F>H AND G>H. H IS TOO SMALL. PUT THE MAIN SUB-EXPRESSIONS OF H AT THE END OF THE LIST PH AND GO TO STEP 3.

5.5 H>F AND H>G. H IS TOO LARGE. PUT THE MAIN SUB-EXPRESSIONS OF H AT THE END OF THE LIST PH AND GO TO STEP 3.

THE LAST TWO STEPS ARE AGAIN BASED ON HEURISTIC RULE 2.

PROOF BY SYNTHESIS

THIS METHOD STARTS WITH TRUE RELATIONS AND COMBINES THEM IN SUCH A WAY THAT WILL HOPEFULLY YIELD THE RELATION TO BE PROVED. FURTHERMORE, THIS METHOD ACTS LIKE A 'PASSIVE COORDINATOR' OF THE PARTIAL SUCCESS OF OTHER METHODS. IN OTHER WORDS, WHILE THE TOP-MOST ROUTINE WOULD BE THE 'ACTIVE COORDINATOR' THAT MAKES POLICY DECISIONS, E.G., WHICH SUB-GOAL TO ATTACT NEXT, WHICH METHOD TO USE, HOW MUCH EFFORT TO SPEND, ETC., THE SYNTHESIS METHOD METABOLIZES THE INCOMPLETE RESULTS OF OTHER METHODS AND TRY TO USE SUCH PARTIAL RESULTS FOR A SYNTHESIS STEP. THIS METHOD IS CALLED EVERYTIME A SUB-GOAL CANNOT BE PROVED. TWO TASKS WILL BE

ACCOMPLISHED: I) AN ORGANIZATION WILL BE MADE OF ALL THE NEW KNOWLEDGE OBTAINED BY THE PREVIOUS FRUITLESS ATTEMPT. II) A TEST WILL BE MADE TO SEE WHETHER A BRIDGE CAN BE MADE BETWEEN NEWLY KNOWN FACTS AND OLD UNSOLVED PROBLEMS. THUS THIS METHOD WOULD AVOID THE EMBARRASSING SITUATION IN WHICH THE PROGRAM ACTUALLY FINDS AND PROVES ALL THE NECESSARY FACTS TO OBTAIN THE MAIN GOAL AND DOES NOT REALIZE IT.

FOR EACH DIFFERENT FRUITLESS ATTEMPT TO PROVE THE MAIN CONJECTURE, A LOGICAL FORMULA IS CONSTRUCTED THAT IS CALLED THE 'LOGICAL DESCRIPTION OF THE ATTEMPT' (LDA). THIS FORMULA IS A GENERALIZATION OF SLAGLE'S [32] AND-OR TREES. FOR EXAMPLE, SUPPOSE THE MAIN PROBLEM IS TO SHOW

$$Q*(A+B) > C+D$$

ONE ATTEMPT MAY BE TO SHOW

$$Q > 1, \quad A+B > 0, \quad A+B > C+D,$$

IN THIS CASE THE LDA WOULD BE THE LOGICAL FORMULA

$$((A>C \wedge B>D) \vee (A>D \wedge B>C)) \wedge Q>1 \wedge A+B>0$$

ANOTHER ATTEMPT COULD BE TO SHOW

$$Q > C, \quad A+B > 1, \quad Q > 0, \quad 0 > D$$

IN THIS CASE THE LDA WOULD BE THE FORMULA

$$Q>C \wedge Q>0 \wedge 0>D \wedge (A>1 \wedge B>0 \vee A>0 \wedge B>1)$$

NOW, THERE IS A LIST CALLED 'SUBPROBLEM' WHICH CONTAINS ALL THE CONJECTURES THAT THE PROGRAM HAS MADE AND FAILED TO PROVE. THE LIST 'SUBPROBLEM' HAS AN ATTRIBUTE CALLED 'LOGICTREES' WHOSE VALUE IS A LIST IN ONE TO ONE CORRESPONDENCE WITH THE LIST OF CONJECTURES. FOR EACH

CONJECTURE, SAY C[1], IN THE LIST 'SUBPROBLEM' THE CORRESPONDING ELEMENT IN THE VALUE LIST OF 'LOGICTREES' IS A LIST WHICH CONTAINS ALL THE LDA'S THAT HAVE AT LEAST ONE OCCURRENCE OF C[1].

LET US SEE HOW THE SYNTHESIS METHOD WORKS. WE HAVE SEEN THAT EVERY METHOD STORES IN 'GREAT' ANY RELATION THAT IT HAPPEN TO PROVE. WE DEFINE A SYNTHESIS STEP AS FOLLOWS: GIVEN A TRUE RELATION Q (I.E. Q IS CONTAINED IN THE LIST 'GREAT') AND A NUMERICALLY TESTED CONJECTURE C (CONTAINED IN THE LIST 'SUBPROBLEM'), THERE IS SYNTHEIS STEP BETWEEN Q AND C IF THERE IS A SUBSTITUTION <S> (C.F. CHAPTER III'S DEFINITIONS 6, 7, 8) SUCH THAT:

1.- C :: Q<S>

2.- EITHER THERE IS NO CONSTRAINT IN THE VARIABLES OF Q, OR THE COMPLETION VALUES OF Q WITH RESPECT TO C SATISFY THE CONSTRAINTS OF THE CORRESPONDING VARIABLES.

IN OTHER WORDS, WE ACCOMPLISH A SYNTHESIS STEP WHENEVER WE MAKE A THEOREM SCHEMA OUT OF A PROVED RELATION. FOR EXAMPLE, SUPPOSE WE HAVE PROVED THE RELATION

$$A\uparrow 2 + B\uparrow 2 > A*B$$

WITH NO CONSTRAINTS ON A AND B, THEN WE COULD PROVE THE CONJECTURE

$$SIN(X)\uparrow 2 + (Y/Z)\uparrow 2 > SIN(X)*(Y/Z)$$

WITH A SIMPLE SUBSTITUTION, I.E., BY MEANS OF THE SYNTHESIS STEP. ON THE OTHER HAND, SUPPOSE WE HAVE THE ASSUMPTIONS:

$$X > 1$$
$$Y > 0$$

AND WE HAVE PROVED THAT

$$LN(X) > 0$$

THEN WE COULD PROVE THAT

$$LN(1+Y) > 0$$

IF WE PROVE FIRST THAT

$$1 + Y > 1$$

WHICH IS THE CONSTRAINT ON X WE HAVE TO FULFILL IN ORDER TO APPLY THE SYNTHESIS STEP.

THIS METHOD CANNOT BE USED WHEN WE ARE PROVING BY INDUCTION SINCE FROM THE INDUCTION HYPOTHESIS WE CAN ALWAYS BY SUBSTITUTION OBTAIN THE RELATION FOR N+1.

THE MAIN STEPS OF AN ALGORITHM WE COULD USE TO IMPLEMENT THE SYNTHESIS METHOD ARE:

1.- CHECK FOR THE EXISTENCE OF A SYNTHESIS STEP, I.E., VERIFY WHETHER A JUST PROVED INEQUALITY Q IA S GENERALIZATION OF AN OLD CONJECTURE; OR VERIFY WHETHER A JUST CREATED CONJECTURE IS A PARTICULARIZATION OF AN OLD PROVED INEQUALITY.

2.- WHEN A SYNTHESIS STEP IS FOUND BETWEEN A CONJECTURE C AND A TRUE RELATION Q WE CONTINUE AS FOLLOWS. WE EVAL EVERY 'LDA' THAT CONTAINS C REPLACING SUCH C BY THE VALUE TRUE. IF SOME 'LDA' COLLAPSES TO TRJE THEN THE MAIN PROBLEM HAS BEEN SOLVED. OTHERWISE, WE UPDATE THE INFORMATION OF THE SYSTEM. THUS WE REMOVE C FROM THE LIST 'SUBPROBLEM' AND APPEND IT TO THE LIST 'GREAT'. ALL THE 'FDA' THAT CONTAIN C ARE SIMPLIFIED BY THEIR EVALUATION AFTER REPLACING C BY TRUE.

THE METHODS OF PROOF BY ANALYSIS, LOGICAL CHAIN, AND TRANSITIVITY HAVE BEEN PROGRAMMED AND TESTED SUCCESSFULLY. THEY ARE

LISTED IN APPENDICES 21, 22 AND 19. RESPECTIVELY. BUT DUE TO THE LIMITATIONS IN THE SYSTEM, THAT WE HAVE ALREADY MENTIONED, THE SYNTHESIS METHOD WAS NOT MACHINE TESTED SINCE IT NEEDS TO BE IN THE MACHINE AT THE SAME TIME AS ALL THE REMAINING METHODS.

SIMPLIFICATION ROUTINES

WE HAVE PROGRAMMED FIVE SIMPLIFICATION ROUTINES THAT CAN BE CALLED FROM ANYWHERE IN THE PROGRAM. THEY COVER MOST CAPABILITIES DESCRIBED IN THE LITERATURE, E.G., [34],[36]. THE MAIN NOVELTY OF THESE ROUTINES LIES IN THE WAY THEY ARE PROGRAMMED AND ORGANIZED. WE PRODUCED VERY LITTLE BEYOND WHAT HAS ALREADY BEEN ACCOMPLISHED BY OTHER WORKERS. EVEN MORE OUR FACTORIZATION ROUTINE IS WEAK. NEVERTHELESS, THESE ROUTINES PROVIDE A RATHER SATISFACTORY ALGEBRAIC SIMPLIFICATION SUPPORT TO THE OTHER TASKS WE ARE INTERESTED IN. ON THE OTHER HAND, WE WERE ABLE TO FORMALLY DEFINE THE CLASS OF FORMULAE THAT CHARACTERIZE THE INPUT AND OUTPUT OF A PROCEDURE. THE FIRST THREE ROUTINES ARE SELF-EXPLANATORY, APPENDICES 8 TO 10, SINCE THEY ARE MARKOV ALGORITHMS AND EACH ONE OF THEIR PRODUCTIONS STATES A RULE OF TRANSFORMATION. THE FIRST ALGORITHM, 'MINUSOUT', ELIMINATES THE UNARY MINUS OF A FORMULA. WE PROVED IN CHAPTER III (STATEMENT 1) THAT THE ALGORITHM TERMINATES. WE ALSO PROVED THAT IT TRANSFORMS ANY FORMULA FROM THE CLASS IG INTO AN ALGEBRAICALLY EQUIVALENT FORMULA IN THE CLASS TG. THE CLASSES IG AND TG ARE DEFINED IN APPENDIX A-2.

THE SECOND ALGORITHM, 'RATIO', TRANSFORMS A FORMULA INTO A

RATIONAL FORM. THIS ALGORTIHM ALSO WAS PROVED IN CHAPTER III (STATEMENT-4). THE THIRD ALGORITHM, 'EXPAND' USES THE DISTRIBUTIVE LAW OF '*' WITH RESPECT TO '+' AND '-', OR IT USES THE DEFINITION OF INTEGER POWER IN TERMS OF MULTIPLICATION TO EXPAND A FORMULA. BY MEANS OF CHAPTER III'S THEOREM III WE CAN PROVE THAT THIS ALGORITHM TERMINATES AND THAT IT TRANSFORMS ANY FORMULA IN THE CLASS QG INTO AN ALGEBRAICALLY EQUIVALENT FORMULA IN THE CLASS RG.

THE TRANSFORMATION PRODUCED IN A FORMULA BY THE PROCEDURE 'COLLECT' (APPENDIX-11) MIGHT BE BETTER DESCRIBED BY MEANS OF FORMAL DEFINITIONS OF THE CLASS OF INPUTS AND THE CLASS OF OUTPUTS. WE HAVE THE FOLLOWING DEFINITIONS:

```
ATOM   ::=   NUMBER | VARIABLE
IGTO   ::=   INTEGER GREATER THAN ONE
   F   ::=   ATOM | ATOM↑IGTO | F*F
 CFP   ::=   NUMBER | VARIABLE | VARIABLE↑IGTO
 NOF   ::=   CFP | CFP*NOF
  CF   ::=   NOF
NOPS   ::=   CF | CF + NOPS
 CPS   ::=   NOPS
  CP   ::=   CPS | CPS - CPS
```

WHERE 'CFP' IS A CANONICAL FACTOR PRIMARY; 'NOF' IS A NON ORDERED FACTOR; 'CF' IS A CANONICAL FACTOR; 'NOPS' IS A NON ORDERED POSITIVE SUM; 'CPS' IS A CANONICAL POSITIVE SUM; AND 'CP' IS A CANONICAL POLYNOMIAL.

THERE IS A LIST CALLED 'LEXICO' WHERE ONE MUST STORE THE NAMES OF ALL THE VARIABLES THAT MIGHT OCCUR IN A FORMULA TO BE SIMPLIFIED. THEIR ORDER OF OCCURRENCE WITHIN THE LIST 'LEXICO' IS BY DEFINITION THE LEXICOGRAPHIC ORDERING. MORE PRECISELY, WE DEFINE LEXICOGRAPHIC ORDER BY MEANS OF THE INTEGER VALUED FUNCTION 'LO'.

$$LO(NUMBER) = 0$$
$$LO(\text{ I TH VARIABLE IN LEXICO}) = \text{I TH PRIME}$$
$$LO(VARIABLE\uparrow INTEGER) = LO(VARIABLE)$$

A CANONICAL FACTOR, CF, IS A NOF THAT HAS NO TWO CFP'S WITH THE SAME LO. FURTHERMORE, SUCH CFP'S ARE ARRANGED IN INCREASING ORDER OF LO. FOR THE NEXT SYNTACTIC DEFINITION WE NEED A SLIGHTLY CHANGED DEFINTION OF LEXICOGRAPHIC ORDER. THE FUNCTION LO' IS DEFINED AS FOLLOWS

$$LO'(ATOM) = LO(ATOM)$$
$$LO'(VARIABLE\uparrow INTEGER) = LO(ATOM)\uparrow INTEGER$$
$$LO'(F1*F2) = LO'(F1)*LO'(F2)$$

A CANONICAL POSITIVE SUM, CPS, IS A NOPS WITH NO TWO TERMS 'CF' HAVING THE SAME LO' AND .FURTHERMORE THE TERMS CF ARE ARRANGED WITH INCREASING LO'. IN A CANONICAL POLYNOMIAL CP, NO TWO TERMS CAN HAVE THE SAME LO'.

THE PROCEDURE 'COLLECT' TRANSFORMS A FORMULA IN THE CLASS 'EP' (APPENDIX A-2) INTO AN ALGEBRAICALLY EQUIVALENT FORMULA IN THE CLASS 'CP'. THE PROCEDURE 'COLLECT' USES THE PROCEDURE 'TERMS' WHICH MAPS ANY FORMULA IN THE CLASS 'F' INTO AN ALGEBRAICALLY EQUIVALENT FORMULA IN THE CLASS 'CF'. FURTHEREMORE WHILE THE PROCEDURE 'TERMS' IS ARRANGING A FORMULA IT COMPUTES TWO NUMBERS THAT DESCRIBE THE FORMULA IN A RATHER PRECISE WAY. SUCH NUMBERS ARE:

C1 = PRODUCT OF THE LO'S OF ALL DISTINCT VARIABLES IN THE CANONICAL FORMULA.

$$C2 = \begin{cases} C3 & \text{IF } C3 < 2\uparrow 42 \\ 11 & \text{OTHERWISE} \end{cases}$$

WHERE

C3 = 2↑C4 + 3↑C5 + 5↑C6

AND

C4 = C7 - C6
C5 = C8 - C6

C6 = MINIMUM EXPONENT OF A VARIABLE IN THE CANONICAL FORMULA.
C7 = MAXIMUM EXPONENT OF A VARIABLE IN THE CANONICAL FORMULA.
C8 = ENTIER(AVERAGE EXPONENT IN THE VARIABLES OF THE CANONICAL FORMULA).

THE INPUT FORMULA FOR THE PROCEDURE 'COLLECT' MUST BE IN THE CLASS 'EP' (APPENDIX A-2). IT IS A SUM OF TERMS EACH OF WHICH IS IN CLASS 'F'. THE FIRST OPERATION OF THE PROCEDURE COLLECT IS TRANSFORMING SUCH TERMS FROM THE CLASS F INTO THE CLASS 'CF', WHICH IS ACCOMPLISHED BY THE PROCEDURE 'TERMS'. AFTER THESE TRANSFORMATIONS HAVE TAKEN PLACE, THE DESCRIPTIVE NUMBERS (C1, C2) HAVE BEEN COMPUTED FOR EACH TERM. IT IS EASILY SEEN THAT FOR TWO TERMS TO BE ALIKE (I.E., TO BE EQUAL EXCEPT FOR A CONSTANT FACTOR) THEY MUST HAVE THE SAME PAIR OF DESCRIPTIVE NUMBERS. THUS THE PROCEDURE USES THESE NUMBERS TO ELIMINATE UNNECESSARY AND EXPENSIVE EQUALITY TESTS. THE PROCEDURE FINALLY COLLECTS ALIKE TERMS AND ARRANGES THEM IN LEXICOGRAPHIC ORDERING.

THE LAST PROCEDURE WE HAVE TO ACCOMPLISH ALGEBRAIC MANIPULATIONS IS THE PROCEDURE 'FACTOR' (APPENDIX-12). IT IS A BOOLEAN PROCEDURE WHICH IS TRUE WHENEVER IT CAN TRANSFORM THE INPUT, WHICH IS A POLYNOMIAL. THIS PROCEDURE TRIES TO ACHIEVE THE FOLLOWING GOALS:

1.- TO FACTOR OUT INTEGER COEFFICIENTS.

2.- TO FACTOR OUT HIGHEST POSSIBLE POWERS OF VARIABLES.

3.- TO TRANSFORM THE POLYNOMIAL INTO A SUM OF SQUARES.

THIS PROCEDURE WAS PROGRAMMED TO AID THE INEQUALITY PROVING ROUTINES AND THE SIGN ANALYSIS ROUTINE. THUS THE THIRD GOAL BECOMES APPARENT, SINCE BY REARRANGING A POLYNOMIAL INTO SUM OF SQUARES WE CAN SHOW IT IS NON-NEGATIVE.

CHAPTER VI. CONCLUSIONS

THIS DISSERTATION HAS ATTEMPTED TO ACCOMPLISH SEVERAL GOALS. FIRST, IT HAS PRESENTED A NOTATION AND A PROGRAMMING LANGUAGE TO BE USED AS DESCRIPTIVE FORMALISM FOR EXPRESSING ALGEBRAIC MANIPULATION ALGORITHMS NATURALLY AND DIRECTLY. IT IS HOPED THAT THIS DESCRIPTIVE FORMALISM HAS BROUGHT TO THE TASK THE ABILITY TO EXHIBIT CLEARLY THE STRUCTURE OF FORMULA MANIPULATION ALGORITHMS. WE HAVE ATTEMPTED TO PUT FORTH A SET OF PRIMITIVE FORMULA MANIPULATION OPERATIONS WHICH CAN BE COMBINED IN A NATURAL WAY IN ORDER TO SYNTHETIZE ALGORITHMS THAT PERFORM MATHEMATICS MECHANICALLY. WHAT IS MEANT BY NATURAL IS THAT THE METHODS AND NOTATIONS THAT ARE COMMONLY USED TO PERFORM ALGEBRAIC MANIPULATIONS ARE EASILY AND DIRECTLY FORMALIZED BY MEANS OF NOTATIONS AVAILABLE IN FORMULA ALGOL. A KEY TO THIS PROCESSES, WE BELIEVE, IS THAT NOTATIONS WHICH ARE USED IN MATHEMATICS TO DENOTE ALGEBRAIC EXPRESSIONS AND TO EXPRESS RULES FOR TRANSFORMING EXPRESSIONS MAY BE USED DIRECTLY IN FORMULA ALGOL, E.G., (-A)*(-B)=A*B. NUMEROUS EXAMPLES HAVE BEEN PRESENTED WHICH ILLUSTRATE THE USE OF FORMULA ALGOL NOTATION AS A TOOL FOR DESCRIBING ALGEBRAIC MANIPULATION ALGORITHMS.

SECOND, WE HAVE USED FORMULA ALGOL TO DESCRIBE A VARIETY OF ALGORITHMS USEFUL IN FORMULA MANIPULATION, AND IN THE ACT OF WRITING THESE ALGORITHMS WE HAVE EXHIBITED IN A READABLE FORM ALGORITHMIC STRUCTURES ADEQUATE TO MECHANIZE THE UNDERLYING MATHEMATICS. THIS IS REVEALING WITH REGARD TO SHOWING IN DETAIL WHAT IS SUFFICIENT TO PERFORM

THE TASK OF MECHANIZATION. THE MARKOV ALGORITHM FEATURE OF THE LANGUAGE, WE THINK, HAS BEEN PARTICULARLY EFFECTIVE IN REVEALING THE STRUCTURES OF ALGEBRAIC TRANSFORMATIONS. THIS CAN BE OBSERVED IN APPENDICES 5, 6, 8, 9, 10, 13, AND 18.

THIRD, WE HAVE EXPLORED MATHEMATICALLY SOME REQUIREMENTS TO BE IMPOSED ON MARKOV ALGORITHMS TO GUARANTEE THAT THEY TERMINATE FOR CERTAIN CLASSES OF INPUTS, AND TO GUARANTEE THAT THE RESULTS OF MARKOV TRANSFORMATIONS AT TERMINATION ARE OF THE REQUIRED FORM. THESE RESULTS ALLOW US TO DETERMINE CONVENIENTLY THAT CERTAIN KINDS OF MARKOV ALGORITHMS PERFORM AS EXPECTED AND THUS TO HAVE SOME CONFIDENCE THAT OUR PROGRAMS ARE OPERATING ACCORDING TO OUR INTENTIONS.

FOURTH, WE HAVE TWO CASE STUDIES (LIMITS AND INEQUALITIES) WHICH ILLUSTRATE HOW THE BASIC FORMULA MANIPULATION CAPABILITIES WE HAVE INTRODUCED CAN BE COMBINED IN PROGRAMS THAT SOLVE A BROAD VARIETY OF PROBLEMS AT A HIGH LEVEL OF SOPHISTICATION. THIS PROVIDES A MEASUREMENT OF WHAT CAN BE ACCOMPLISHED USING THE NOTATION AND ALGORITHMS WE HAVE INTRODUCED.

THE MAIN LIMITATION THAT WE FIND IN OUR RESULTS LIES IN THE SIZE OF OUR RUNNING EXAMPLES.

NOTWITHSTANDING THE LARGE NUMBER OF SUCCESSFUL PROGRAMS THAT HAVE BEEN WRITEN IN FORMULA ALGOL, IT IS CLEAR THAT IMPROVEMENTS ARE NEEDED. THESE IMPROVEMENTS MUST BE IN THE DESIGN OF THE LANGUAGE ITSELF AS WELL AS IN ITS IMPLEMENTATION. THE PRESENT RUNNING SYSTEM WAS

IMPLEMENTED [16], [17] USING THE FORMAL COMPILER SYSTEM 'FS_' [38]. THIS SYSTEM WAS EXTREMELY USEFUL IN THE DESIGN STAGES, SINCE WITH ALMOST NO EFFORT WE COULD MODIFY THE SYNTAX, AS WELL AS THE SEMANTICS, OF THE DIVERSE PARTS OF THE LANGUAGE. BUT ONCE THE DESIGN WAS ACCEPTED AND WE FROZE THE SYSTEM SO WE COULD START TO RUN NON-TEST PROGRAMS, WE BEGAN TO PAY A PRICE FOR THE FLEXIBILITY USED AT THE DESIGN STAGES. THE FS_ SYSTEM IMPOSED SEVERAL LIMITATIONS IN VARIOUS TABLE SIZES AND DID NOT PROVIDE AN EFFICIENT OPERATIONAL SYSTEM. AMONG OTHER PROBLEMS, WE HAD A LIMIT OF 7000_{10} WORDS FOR COMPILING CODE AND WE COULD NOT MOVE PROCEDURES BACK AND FORTH TO THIS SPACE DURING THE EXECUTION OF A PROGRAM. FOR THESE REASONS WE LACK A COLLECTION OF COMPLETE EXAMPLES THAT ILLUSTRATE THE COMBINED POWER OF ALL THE METHODS AND TECHNIQUES THAT WE HAVE PROGRAMMED.

ALTHOUGH THE TECHNOLOGY TO CORRECT THIS LIMITATIONS HAS BEEN KNOWN FOR SOME TIME, OUR FIRST CONCERN WAS NOT TO PRODUCE A SYSTEM FOR USERS. OUR MAIN GOAL WAS TO EXPLORE NEW FORMS FOR DESCRIBING SYMBOLIC COMPUTATIONS. HOWEVER, ONE CANNOT EFFECTIVELY DESIGN A PROGRAMMING LANGUAGE WITHOUT A RUNNING TRANSLATOR WITH WHICH TO EXPERIMENT. THE SYSTEM WE HAVE PRODUCED LACKS THE CONVENIENCES OF A FULL OPERATIONAL SYSTEM, E.G., GOOD INPUT-OUTPUT FACILITIES, RELOCATABLE CODE, LIBRARY OF PROCEDURES, ETC. YET, USERS CAN RUN PROGRAMS AND OBTAIN RESULTS USING THE POWERFUL FEATURES FOR FORMULA MANIPULATION AND LIST PROCESSING DESCRIBED IN [28]. IN PARTICULAR, THE RUNNING SYSTEM HAS A SUPERB RECURSIVE MECHANISM CONTROL, CLEVERLY IMPLEMENTED BY MR. R. KRUTAR. THERE IS A PROJECT NOW UNDER WAY TO IMPLEMENT A CONVERSATIONAL FORMULA

ALGOL IN THE NEW CARNEGIE TECH IBM 360-67 MACHINE.

THE DESIGN OF THE LANGUAGE ITSELF CAN AND MUST BE IMPROVED. THE LANGUAGE EVOLVED AS DIRECT CONSEQUENCE OF EXPERIMENTATION, THUS IT HAS A HETEROGENEOUS STRUCTURE. IT IS THE AUTHOR'S OPINION THAT IS VERY DIFFICULT, IF NOT IMPOSSIBLE, TO FORESEE ALL THE NECESSARY INGREDIENTS THAT A LANGUAGE LIKE FORMULA ALGOL SHOULD HAVE SO THAT ONE COULD COMBINE SUCH INGREDIENTS FROM THE VERY BEGINNING IN AN ELEGANT AND HOMOGENEOUS MANNER. IT IS ONLY NOW WITH THE EXPERIENCE WHICH HAS BEEN GAINED THAT WE CAN GO BACK AND ANALYZE THE VARIOUS PARTS OF THE LANGUAGE AND INTEGRATE THEM INTO A MORE HOMOGENEOUS STRUCTURE.

DURING THE DESIGN OF FORMULA ALGOL WE HAVE ISOLATED CONCEPTS THAT WERE IMPLICIT IN ALGEBRAIC MANIPULATIONS, AND WE ATTACHED SUCH CONCEPTS TO PRIMITIVES OF THE LANGUAGE. FOR EXAMPLE, BY MEANS OF DOT-PROCEDURES ONE CAN INTRODUCE ARBITRARY FUNCTIONS; FURTHEREMORE, ONE CAN USE THESE FUNCTIONS NOT ONLY AS PART OF FORMULAE, BUT ALSO GIVE ALGORITHMIC DESCRIPTIONS OF THE BEHAVOIR OF SUCH FUNCTIONS UNDER VARIOUS MATHEMATICAL TASKS, SUCH AS DIFFERENTIATION, EVALUATION, SIMPLIFICATION, ETC. ANOTHER EXAMPLE IS GIVEN BY THE PREDICATE 'CONTAINS', >>, WHICH WE SAW CAN BE USED TO 'FOCUS ONE'S ATTENTION' AT SUB-EXPRESSIONS WITH GIVEN CHARACTERISTICS; ALSO THIS PREDICATE CAN BE USED TO PRUNE IRRELEVANT SUB-EXPRESSIONS BY MEANS OF ANTI-SUBSTITUTION. ANOTHER IMPORTANT FEATURE BORROWED FROM MATHEMATICS IS THAT THE MEANING OF ARITHMETIC OR BOOLEAN EXPRESSIONS DEPENDS NOT ONLY ON SYNTAX, BUT ALSO UPON TYPE DECLARATIONS, I.E., DEFINITIONS. WITH THIS FEATURE WE GAINED DESCRIPTIVE POWER WITHOUT

INCREASING THE NUMBER OF SYNTACTIC CONSTRUCTIONS, E.G., THE PROGRAMS IN APPENDIX-1.

A GREAT MANY ALGEBRAIC MANIPULATION PROBLEMS CAN BE MECHANIZED BY MEANS OF A RECURSIVE TOP-DOWN ALGORITHMIC CONTROL, E.G., DIFFERENTIATION, NUMERIC EVALUATION, SUBSTITUTION, EVALUATION OF TRIVIAL LIMITS, THE ANALYSIS METHOD FOR PROVING INEQUALITIES, THE 'TESTSIGN' PROCEDURE, ETC. ALL THESE TASKS HAVE A COMMON CONTROL STRUCTURE:

1.- CHECK IF THE INPUT HAS ONE OF A GIVEN COLLECTION OF PROPERTIES (ONE SUCH PROPERTY IS ALWAYS THAT OF BEING ATOMIC). IF ONE OT THE PROPERTIES IS FOUND AN EXPLICIT SOLUTION IS GIVEN AND THE ALGORITHM EXITS. OTHERWISE

2.- THE MAIN OPERATOR AND THE MAIN SUB-EXPRESSIONS OF THE INPUT ARE SEPARATED. THE ALGORITHM IS APPLIED RECURSIVELY TO EACH ONE OF THE SUB-EXPRESSIONS AND A RESULT IS CONSTRUCTED USING THE MAIN OPERATOR AND THE RESULTS PRODUCED BY PROCESSING THE SUB-EXPRESSIONS.

THIS SUGGEST VERY STRONGLY THE CREATION OF A GENERAL PURPOSE SEQUENCER THAT WILL EMBRACE ALL THE BOOKKEEPING AND CONTROL. THIS GENERAL PURPOSE SEQUENCER SHOULD BE PART OF THE LANGUAGE, ITS FUNCTION WILL BE TO SEQUENCE THROUGH ANY WELL DEFINED DATA STRUCTURE AND TO APPLY, AT EACH STEP, TESTS THAT EITHER STOP THE SEARCH OR SELECT THE APPROPRIATE ACTION. THE USER WILL HAVE TO PROVIDE ONLY THE ESSENCE OF HIS ALGORITHM, I.E., AN ALGORITHMIC DESCRIPTION OF THE TESTS AND THE ACTIONS.

WE HAVE INVESTIGATED THE POSSIBILITIES OF MECHANIZATION OF LIMITS PROCESSES AND PROOFS OF INEQUALITIES. IN DOING THIS WE HAVE POINTED OUT NEW PROBLEMS, TO WHICH WE HAVE GIVEN PARTIAL SOLUTIONS. BECAUSE OF THE BOTTLENECK PRODUCED BY THE MAXIMUM SIZE FOR COMPILED CODE

WE DID NOT HAVE THE OPPORTUNITY TO PROGRAM AND TEST 'INTELLIGENT' CONTROL MECHANISMS. THE REAL TEST OF USEFULNES FOR ALL THE METHODS WE HAVE DISCUSSED WILL COME WHEN WE CAN RUN ALL OF THEM UNDER ONE SOPHISTICATED CONTROL MECHANISM. IN PARTICULAR, IT IS OUR VIEW THAT LEARNING DEVICES WILL BE EXTREMELY USEFUL IN SHORTENING THE EXECUTION TIME, MAINLY IN REPETITIVE SITUATIONS WHICH MAY BE CREATED, FOR INSTANCE, BY THE HEURISTIC I (CHAPTER IV). THIS HEURISTIC PROPOSES A NOT-TOO-CAREFUL PROCESSING OF THE WHOLE PROBLEM, AND ONLY IF THERE IS A FAILURE AT THE VERY END, RETURN AND TRY AGAIN WITH MORE CARE. THE PROGRAMS ALSO SHOULD REMEMBER THE ASSUMPTIONS THEY HAVE MADE SO THEY CAN DETECT WHEN THEY ARE GENERATING A CONTRADICTORY ASSUMPTION.

THE RESEARCH WE HAVE BEGUN ON LIMITS AND INEQUALITIES, EVEN THOUGH IT IS NOT COMPLETE, PROVIDES A MISSING LINK BETWEEN THE PRESENT STATUS OF ALGEBRAIC MANIPULATION AND THE TASK OF MECHANIZING THE CONVERGENCE TESTS FOR MATHEMATICAL SERIES. USING OUR RESULTS WE CAN DESCRIBE WITH EASE A MECHANIZATION FOR SOLVING THE FOLLOWING PROBLEMS: GIVEN A SEQUENCE $\langle A[N]\rangle$, 1) VERIFY WHETHER FOR SUFFICIENTLY LARGE N, $A[N]>0$; 2) TEST IF $A[N]$ IS MONOTONIC INCREASING OR DECREASING; 3) TEST IF FOR SUFFICIENTLY LARGE N $A[N]>B[N]$, WHERE $\langle B[N]\rangle$ IS ANOTHER GIVEN SEQUENCE; 4) FIND THE LIMIT OF $A[N]$ WHEN $N\rightarrow INFI+$. ALL THESE PROBLEMS ARE CONDITIONS FOR THE WELL KNOWN COVERGENCE TESTS, E.G., THE RATIO TEST, THE ROOT TEST, THE COMPARISON TEST, ETC. TO SOLVE THE CONVERGENCE PROBLEM ONE ALSO NEEDS THE ABILITY TO INTEGRATE FORMALLY WHICH HAS BEEN DISCUSSED ELSEWHERE [32], [22].

THE NOTATION USED IN THE VARIOUS ALGORITHMS PRESENTED HERE, THE ALGORITHMS THEMSELVES, THEIR PROOFS OF TERMINATION, THE CHARACTERIZATION OF THEIR INPUTS AND OUTPUTS, AND THE STUDIES FOR MECHANIZING LIMITING PROCESSES AND INEQUALITIES, TAKEN TOGETHER, CONSTITUTE AN APPROACH TO THE BASIC GOAL OF MECHANIZING PORTIONS OF ALGEBRA AND ANALYSIS. IT IS CERTAINLY NOT THE ONLY APPROACH AND OTHERS CAN AND SHOULD BE EXPLORED AND COMPARED WITH IT. HOWEVER WE ARE HOPEFUL THAT THE METHODS AND TECHNIQUES INTRODUCED IN THIS APPROACH WILL CONTRIBUTE TO SYSTEMS AT EVEN HIGHER LEVELS OF PERFORMANCE HAVING THE POWER TO FUNCTION AS PRACTICAL MATHEMATICAL ASSISTANTS.

APPENDIX A-1, THE REPLACEMENT THEOREM

WE ARE GOING TO USE SOME CONCEPTS INTRODUCED EARLIER IN THIS DISSERTATION; THUS WE SUGGEST THAT THE READER SHOULD CHECK IN CHAPTER III: DEFINITION-2, OF 'PATH'; DEFINITION-6, OF IDENTITY; DEFINTIONS 7,8,9,10 AND 11 RELATED TO THE CONCEPT OF SUBSTITUTION.

LET F AND G BE TWO FORMULAE DEFINED AS FOLLOWS:

F :: R<S> WITH <S> = < X[I]:A[I];N >

G :: R<E> WITH <E> = < X[I]:B[I];N >

I.E. F AND G ARE PARTICULARIZATIONS OF THE SAME FORMULA, R. F IS OBTAINED FROM R BY SUBSTITUTION OF THE VARIABLES X[I] FOR THE FORMULAE A[I] WHILE G IS OBTAINED FROM R BY SUBSTITUTION OF THE VARIABLES X[I] FOR THE FORMULAE B[I]. WE ARE GOING TO STATE AND PROVE THE CHARACTERISTICS THAT A FORMULA R SHOULD HAVE IN ORDER THAT F AND G BE ASYMPTOTICALLY EQUIVALENT WHENEVER FOR I=1,..,N A[I] AND B[I] ARE ASYMPTOTICALLY EQUIVALENT.

THEOREM 2. (REPLACEMENT THEOREM)

LET F AND G BE AS DESCRIBED ABOVE. IF FOR EACH I=1, .., N IT IS THE CASE THAT:

1.- A[I] =INFI= B[I].

2.- LET (O[1], O[2], .., O[K]) BE THE PATH OF VARIABLE X[I], EACH OPERATOR O[J] SHOULD BE RESTRICTED AS FOLLOWS:

2.1 NO O[J] IS THE OPERATOR 'EXP'.

2.2 IF O[J] IS '+' THEN X[I] MUST BE CONTAINED IN THE LEFT BRANCH OF THE OPERATOR O[J] AND THE LIMIT OF THE RIGHT BRANCH MUST EXIST AND IT SHOULD

BE FINITE.

2.3 IF O[J] IS '+'; LET T DENOTE THE BRANCH OF + THAT DOES NOT CONTAIN X[I]; THEN WE REQUIRE THAT:

LIM T/A[I] MUST EXIST AND SHOULD BE FINITE BUT NOT -1, OR

LIM T/B[I] MUST EXIST AND SHOULD BE FINITE BUT NOT -1.

2.4 IF O[J] IS - (BINARY) REPLACE IT WITH +(-UNARY) AND CHECK POINT 2.3.

2.5 IF O[J] IS 'LN' THEN IF LIM A[I] IS 0 OR INFI+ OR 1 THEN IT IS REQUIRED THAT LIM A[I]'/B[I]' = 1 WHERE QUOTE ' MEANS DIFFERENTIATION.

2.6 IF O[J] IS 'SIN' OR 'COS' THEN LIM A[I] MUST BE FINITE AND FURTHERMORE IF SUCH A LIMIT IS A MULTIPLE OF PI/2 THEN IT IS REQUIRED ALSO THAT LIM A[I]'/B[I]' = 1.

THEN:

F =INFI= G.

PROOF:

IT SUFFICES TO PROVE THE THEOREM ASSUMING THAT THE SUBSTITUTIONS <S> AND <E> CONTAIN ONLY ONE SUBSTITUTION PAIR, I.E. ASSUMING THAT <S> = <X:A> AND <E> = <X:B>. BECAUSE, IF WE HAVE PROVED SUCH A CASE, THE SEQUENCE OF FORMULAE:

F[0] = F
F[1] IS OBTAINED FROM F[0] REPLACING A[1] BY B[1].
F[2] IS OBTAINED FROM F[1] REPLACING A[2] BY B[2].
....
F[N] IS OBTAINED FROM F[N-1] REPLACING A[N] BY B[N].

IS SUCH THAT A[J] =INFI= B[J] IMPLIES F[J] =INFI= F[J-1], BUT FROM CHAPTER IV'S COROLLARY-2 OF DEFINITION-3 WE HAVE THAT =INFI= IS AN EQUIVALENCE RELATION, THUS F[0] =INFI= F[N]. BY DEFINITION F[0] IS F AND BY CONSTRUCTION F[N] IS G. QED.

NOW LET US PROVE THE THEOREM FOR THE CASE WHERE <S> AND <E> HAVE ONLY ONE SUBSTITUTION PAIR.

WE ARE GOING TO PROVE IT BY INDUCTION ON K, THE LEVEL OF X IN R.

BASIS. K=1, I.E. R :: X THUS F :: A AND G :: B, THEN TRIVIALLY:

A =INFI= B IMPLIES F =INFI= G.

INDUCTION STEP. SUPPOSE X IS AT LEVEL K+1, THERE IS A FORMULA

' X <OP> T ' OR ' T <OP> X ' OR ' <OP> X '

AT LEVEL K IN R. WE MUST SHOW THAT FOR ALL OPERATORS ALLOWED BY THE CONDITIONS OF THE THEOREM IT IS THE CASE THAT:

A =INFI= B IMPLIES A <OP> T =INFI= B <OP> T, OR
T <OP> A =INFI= T <OP> B, OR
<OP> A =INFI= <OP> B.

NOW WE SHALL PROVE THE ABOVE IMPLICATIONS FOR EACH OPERATOR <OP>. LET Q DENOTE THE BRANCH CONTAINING 'A' AND LET Q' DENOTE THE SAME BRANCH BUT WITH 'A' REPLACED BY 'B'.

CASE <OP> IS '+'.

Q = A+T OR Q = T+A THEN Q' = B+T OR Q' = T+B. BY HYPOTHESIS AT LEAST ONE OF THE FOLLOWING LIMITS EXIST

LIM T/A OR LIM T/B

AND FURTHERMORE IT IS FINITE BUT NOT -1; THEN THE QUOTIENT:

$$(A+T)/(B+T) = \begin{matrix}(1+T/A)/(B/A+T/A)\\ \text{OR}\\ (A/B+T/B)/(1+T/B)\end{matrix}$$

THUS LIM Q/Q' = 1.

CASE <OP> IS '*'.

Q = A*T OR Q = T*A, THEN Q' = B*T OR Q' = T*B. Q/Q' = A*T/B*T = A/B. THUS LIM Q/Q' = LIM A/B = 1.

CASE <OP> IS '/'.

Q = A/T OR Q = T/A, THEN Q' = B/T OR Q' = T/B. Q/Q' = (A/T)/(B/T) = A/B, OR Q/Q' = (T/A)/(T/B) = B/A. THUS LIM Q/Q' = LIM A/B = 1 OR LIM Q/Q' = LIM 1/(A/B) = 1/1 = 1.

CASE <OP> IS '↑'.

BY HYPOTHESIS THE ONLY POSSIBLE CASE IS WHEN Q = A↑T AND LIM T EXISTS AND IS FINITE. THEN Q/Q' = A↑T/B↑T = (A/B)↑T; THUS LIM(Q/Q') = LIM(A/B)↑LIM(T) = 1.

CASE <OP> IS '-' UNARY.

Q = -A, Q' = -B; Q/Q' = A/B; LIM Q/Q' = LIM A/B = 1.

CASE <OP> IS 'ABS'.

Q = ABS(A), Q' = ABS(B); Q/Q' = ABS(A)/ABS(B) = ABS(A/B); THUS LIM Q/Q' = LIM ABS(A/B) = ABS(LIM A/B) = ABS(1) = 1.

CASE <OP> IS 'LN'

Q = LN(A) AND Q' = LN(B) THEN A/B = EXP(LN(A) - LN(B)) I.E. A/B = EXP(Q - Q') AND LIM A/B = 1 IMPLIES LIM Q-Q' = 0 IF LIM Q IS FINITE AND NON ZERO THEN 0 = LIM(Q-Q') = LIM[(1-Q'/Q)*Q] = LIM[1 - Q'/Q]*LIM[Q] = 1 - LIM[Q'/Q] THUS LIM Q/Q' = 1. BY CONTRADICTION WE SEE THAT LIM(Q-Q')=0 IMPLIES LIM(Q)=LIM(Q') THUS FOR THE CASE LIM(Q) IS NON FINITE OR ZERO WE CAN USE L'HOSPITAL RULE: LIM[Q/Q'] = LIM[LN(A)/LN(B)] = LIM[(A'/A)/(B'/B)] = LIM[B/A]*LIM[A'/B'] = 1*1 = 1.

CASE <OP> IS 'SIN'

```
Q/Q' = SIN(A)/SIN(B)
     = SIN(A)*COS(B)/[ SIN(B)*COS(B) ]
     = [ [1/2]*SIN(A+B) - [1/2]*SIN(A-B) ]/[ [1/2]*SIN(2*B) ]
     = [ SIN(A+B) - SIN(A-B) ]/SIN(2*B)
```

NOW, WE HAVE THAT LIM[A]=LIM[B] AND THAT 'SIN' IS A CONTINUOUS FUNCTION. WHEN LIM[SIN(2*B)] ≠ 0, I.E. WHEN LIM[B] IS NOT A MULTIPLE OF PI/2.

```
LIM[ Q/Q' ] = [SIN(LIM[A]+LIM[B]) - SIN(LIM[A]-LIM[B])]/SIN(2*LIM[B])
            = SIN(2*LIM[B])/SIN(2*LIM[B])
            = 1.
```

NOW SUPPOSE LIM[SIN(2*B)] = 0.

LIM[Q/Q'] IS OF THE FORM 0/0 THUS WE CAN APPLY L'HOSPITAL'S RULE:

$$
\begin{aligned}
LIM[Q/Q'] &= LIM[\,[(A'+B')*COS(A+B) - (A'-B')*COS(A-B)\,]/[\,2*B'*COS(2*B)\,] \\
&= LIM[\,(A'/B'+1)*COS(A+B) - (A'/B'-1)*COS(A-B)\,]/[\,2*COS(2*B)\,] \\
&= 2*COS(2*LIM[B])/[\,2*COS(2*LIM[B])\,] \\
&= 1
\end{aligned}
$$

APPENDIX A-2

```
AR[ 10A ]10  ::=  AR[ 10A ]10 + AR[ 10A ]10   |   AR[ 10A ]10 - AR[ 10A ]10   |
                  AR[ 10A ]10 * AR[ 10A ]10   |   AR[ 10A ]10 / AR[ 10A ]10   |
                  AR[ 10A ]10 ↑ AR[ 10A ]10   |             - AR[ 10A ]10     |
                       LN(  AR[ 10A ]10 )     |       EXP(  AR[ 10A ]10 )     |
                     SIN(   AR[ 10A ]10 )     |       COS(  AR[ 10A ]10 )     |
                  ARCTAN(   AR[ 10A ]10 )     |       ABS(  AR[ 10A ]10 )     |
                  ENTIER(   AR[ 10A ]10 )     |       SIGN( AR[ 10A ]10 )     |
                    SQRT(   AR[ 10A ]10 )

RE[ 10A ]10  ::=  AR[ 10A ]10  = AR[ 10A ]10  |   AR[ 10A ]10  > AR[ 10A ]10  |
                  AR[ 10A ]10  < AR[ 10A ]10  |   AR[ 10A ]10 ¬= AR[ 10A ]10  |
                  AR[ 10A ]10 ¬> AR[ 10A ]10  |   AR[ 10A ]10 ¬< AR[ 10A ]10

X[ 10A ]10  ::=  A  | AR[ 10 X[ 10A ]10 ]10
Y[ 10A ]10  ::=  X[ 10A ]10  | RE[ 10 X[ 10A ]10 ]10

ATOM  ::=  NUMBER | VARIABLE
I     ::=  ATOM | I+I | I-I | I*I | I/I | I↑INTEGER | -I
N     ::=  ATOM | N+N | N-N | N*N | N/N | N↑INTEGER
T     ::=  N | -N
GP    ::=  ATOM | GP+GP | GP-GP | GP*GP | GP↑IGTO
IGTO  ::=  <INTEGER GREATER THAN ONE>
Q     ::=  GP | GP/GP | -GP | -(GP/GP)
F     ::=  ATOM | VARIABLE↑IGTO | F*F
S     ::=  F | S+S
EP    ::=  S | S-S
R     ::=  EP | EP/EP | -S | -(S/S)

Z     ::=  Y[ 10ATOM ]10
IG    ::=  Y[ 10I ]10
TG    ::=  Y[ 10T ]10
QG    ::=  Y[ 10Q ]10
RG    ::=  Y[ 10R ]10
```

BNF DEFINITIONS TO CHARACTERIZE THE
FORMULAE USED BY SEVERAL ALGORITHMS

BIBLIOGRAPHY

A VERY EXTENSIVE BIBLIOGRAPHY HAS BEEN COMPILED BY JEAN E. SAMMET. IT IS PUBLISHED IN THE JULY-AUGUST ISSUE OF COMPUTING REVIEWS UNDER THE TITLE: 'AN ANNOTATED DESCRIPTOR BASED BIBLIOGRAPHY ON THE USE OF COMPUTERS FOR NON-NUMERICAL MATHEMATICS'. AN UPDATING LIST HAS BEEN PUBLISHED IN THE SIC-SAM BULLETIN NO. 5, DECEMBER 1966.

1. ABRAHAMS, P., ET AL. 'THE LISP 2 PROGRAMMING LANGUAGE AND SYSTEM'; PROC. FJCC, NOVEMBER 1966.

2. BACKUS, J. W., 'THE SYNTAX AND SEMANTICS OF THE PROPOSED INTERNATIONAL LANGUAGE OF THE ZURICH ACM-GAMM CONFERENCE'; ICIP PARIS, 1959.

3. BECKENBACH, E. AND R. BELLMAN, 'AN INTRODUCTION TO INEQUALITIES'; RANDOM HOSE, 1961.

4. BERKELEY, E. C., AND D. G. BOBROW (EDITORS), 'THE PROGRAMMING LANGUAGE LISP: ITS OPERATION AND APPLICATIONS'; INFORMATION INTERNATIONAL, INC., CAMBRIDGE, MASS.

5. BOBROW, D. G., 'NATURAL LANGUAGE INPUT FOR A COMPUTER PROBLEM SOLVING SYSTEM'; PH.D. THESIS, MIT, PROJ. MAC TECHNICAL REPORT MAC-TR-2, 1964.

6. BROWN, W. S., 'A LANGUAGE AND A SYSTEM FOR SYMBOLIC ALGEBRA ON A DIGITAL COMPUTER'; BELL TELEPHONE LABORATORIES, MURRAY HILL, N.J., 1965.

7. CARACCIOLO DI FLORINO, A., 'PANON-1B A PROGRAMMING LANGUAGE FOR SYMBOLIC MANIPULATION'; UNIVERSITY OF PISA, 1966.

8. CHRISTENSEN, C., 'AMBIT A PROGRAMMING LANGUAGE FOR ALGEBRAIC SYMBOL MANIPULATION'; COMPUTER ASSOCIATES, INC., REPORT NO. CA-64-4-R, OCTOBER 1964.

9. CHURCH, A., 'THE CALCULI OF LAMBDA-CONVERSION'; PRINCETON, SECOND

PRINTING, 1951.

10. COMIT (WITHOUT AUTHOR); 'COMIT PROGRAMMER'S REFERENCE MANUAL'; THE RESEARCH LABORATORY OF ELECTRONICS AND THE COMPUTATION CENTER, MIT, CAMBRIDGE, MASS.

11. ENGLEMAN, C., 'MATHLAB A PROGRAM FOR ON-LINE MACHINE ASSISTANCE IN SYMBOLIC COMPUTATIONS'; PROC. FJCC, NOVEMBER 1965.

12. FENICHEL, R. R., 'AN ON-LINE SYSTEM FOR ALGEBRAIC MANIPULATION'; PH.D. THESIS, HARVARD UNIVERSITY, CAMBRIDGE, MASS., 1966.

13. FLOYD, R. W., 'A DESCRIPTIVE LANGUAGE FOR SYMBOL MANIPULATION'; JOURNAL ACM, VOL 18, 1961.

14. GELERNTER, H., ET AL., 'A FORTRAN-COMPILED LIST-PROCESSING LANGUAGE'; JOURNAL ACM, VOL 7, 1960.

15. GELERNTER, H., ET AL., 'EMPIRICAL EXPLORATIONS OF THE GEOMETRY THEOREM MACHINE'; PROC. WJCC, MAY 1960.

16. ITURRIAGA, R., STANDISH, T., KRUTAR, R., AND J. EARLY, 'TECHNIQUES AND ADVANTAGES OF USING THE FORMAL COMPILER WRITING SYSTEM FSL TO IMPLEMENT A FORMULA ALGOL COMPILER'; PROC. SJCC, MAY 1965.

17. ITURRIAGA, R., STANDISH, T., KRUTAR, R., AND J. EARLY, 'THE IMPLEMENTATION OF FORMULA ALGOL IN FSL'; COMPUTATION CENTER, CARNEGIE INSTITUTE OF TECHNOLOGY, OCTOBER 1966.

18. KNUTH, D., 'BACKUS NORMAL FORM VS. BACKUS-NAUR FORM, LETTER TO THE EDITOR, COMM. ACM, DECEMBER 1964.

19. MARKOV, A. A., 'THEORY OF ALGORITHMS'; ACADEMY OF SCIENCES OF THE USSR, 1954. PUBLISHED IN ENGLISH BY THE NATIONAL SCIENCE FOUNDATION, WASHINGTON D.C. AND THE DEPARTMENT OF COMMERCE.

21. MC CARTHY, J., ET AL., 'LISP I PROGRAMMER'S MANUAL'; COMPUTATION CENTER AND RESEARCH LABORATORY OF ELECTRONICS, MIT, CAMBRIDGE, MASS., MARCH 1960.

22. MOSES, J., 'SYMBOLIC INTEGRATION'; ARTIFICIAL INTELLIGENCE PROJECT, MEMORANDUM NO. 97, MIT, PROJECT MAC, JUNE 1966.

23. NAUR, P., ED., 'REVISED REPORT ON THE ALGORITHM LANGUAGE ALGOL 60'; COMM. ACM, VOL. 6, JANUARY 1963.

24. NEWELL, A., ET AL., 'IPL-V PROGRAMMING MANUAL', SECOND EDITION, PRENTICE HALL, ENGLEWOOD CLIFFS N.J., 1964.

25. NEWELL A., SHAW, J. C., AND H. A. SIMON, 'EMPIRICAL EXPLORATIONS OF

THE LOGIC THEORY MACHINE'; PROC WJCC, FEBRUARY 1957.

26. PERLIS, A. J., AND R. ITURRIAGA, 'AN EXTENSION TO ALGOL FOR MANIPULATING FORMULAE'; COMM. ACM, VOL 7, FEBRUARY 1964.

27. PERLIS, A. J., ITURRIAGA, R., AND T. STANDISH, 'A PRELIMINARY SKETCH OF FORMULA ALGOL'; COMPUTATION CENTER, CARNEGIE INSTITUTE OF TECHNOLOGY, JULY 1965.

28. PERLIS, A. J., ITURRIAGA, R., AND T. STANDISH, 'A DEFINITION OF FORMULA ALGOL'; COMPUTATION CENTER, CARNEGIE INSTITUTE OF TECHNOLOGY, MARCH 1966.

29. RICHARDSON, D., UNTITLED AND UNDATED MIMEOGRAPHEAD DOCUMENT.

30. ROBINSON, J. A., 'A MACHINE-ORIENTED LOGIC BASED ON THE RESOLUTION PRINCIPLE'; JOURNAL ACM, VOL 12, JANUARY 1965.

31. SAMMET, J.E. AND E. BOND, 'INTRODUCTION TO FORMAC'; IEEE TRANSC. VOL. EC-13, AUGUST 1964.

32. SLAGLE, J. R., 'A HEURISTIC PROGRAM THAT SOLVES SYMBOLIC INTEGRATION PROBLEMS IN FRESHMAN CALCULUS'; JOURNAL ACM, VOL. 10, OCTOBER 1963.

33. SPIEGEL, M. R., 'THEORY AND PROBLEMS OF ADVANCED CALCULUS'; SCHAUM PUBLISHING CO., NEW YORK, 1963.

34. TOBEY, R. G., ET AL., 'AUTOMATIC SIMPLIFICATION IN FORMAC'; PROC. FJCC, NOVEMBER 1965.

35. WANG, H., 'TOWARD MECHANICAL MATHEMATICS'; IBM JOURNAL OF RESEARCH AND DEVELOPMENT, VOL 4, JANUARY 1960.

36. WOOLDRIDGE, D., 'AN ALGEBRAIC SIMPLIFICATION PROGRAM IN LISP'; ARTIFICIAL INTELLIGENCE PROJECT, STANFORD UNIVERSITY, MEMORANDUM NO. 11, DECEMBER 1963.

37. CAVINESS, B. F., 'ON CANNONICAL FORMS AND SIMPLIFICATION'; PH.D. THESIS, CARNEGIE INSTITUTE OF TECHNOLOGY, 1967.

APPENDIX 1

TWO PROCEDURES TO INVERT MATRICES, THE FIRST OPERATES WITH NUMBERS, THE SECOND WITH FORMULAE

```
198:          SN        PAGE
199:    15533     PROCEDURE NUMBINVERT(A,B,N); VALUE N; INTEGER N; REAL ARRAY A,B;
200:              BEGIN
201:    15541     REAL X; INTEGER I,J,K;
202:    15543     FOR I←1 STEP 1 UNTIL N DO BEGIN
203:    15573     B[I,I] ← 1/A[I,I];
204:    15623     FOR J←I-1 STEP -1 UNTIL 1 DO BEGIN
205:    15654     X←0;
206:    15657     FOR K←J STEP 1 UNTIL I-1 DO
207:    15710     X ← X - A[I,K]*B[K,J];
208:    15743     B[I,J] ← X*B[I,I] END END END;
209:
210:
211:    16000     PROCEDURE FORMINVERT(A,B,N); VALUE N; INTEGER N; FORM ARRAY A,B;
212:              BEGIN
213:    16006     FORM X; INTEGER I,J,K;
214:    16012     FOR I←1 STEP 1 UNTIL N DO BEGIN
215:    16042     B[I,I] ← 1/A[I,I];
216:    16076     FOR J←I-1 STEP -1 UNTIL 1 DO BEGIN
217:    16127     X←0;
218:    16133     FOR K←J STEP 1 UNTIL I-1 DO
219:    16164     X ← X- A[I,K]*B[K,J];
220:    16225     B[I,J] ← X*B[I,I] END END END;


280:              BEGIN
281:    20707     REAL ARRAY A, B[1:3,1:3];
282:    20732     FORM ARRAY U, V[1:3,1:3];
283:    20754     FORM X; INTEGER I,J;
284:
285:    20756     U[1,2] ← U[1,3] ← U[2,3] ← 0;
286:    21020     U[1,1] ← X; U[2,1] ← 1; U[2,2] ← 2*X + X + 3;
287:    21100     U[3,1] ← 1 + X; U[3,2] ← 1 + (2+X)*X;
288:    21151     U[3,3] ← 3*X;
289:    21171     FORMINVERT( U,V,3);
290:    21200     A[1,2] ← A[1,3] ← A[2,3] ← 0;
291:    21237     A[1,1] ← 5; A[2,1] ← 1; A[2,2] ← 2*5 + 5 + 3;
292:    21303     A[3,1] ← 1 + 5; A[3,2] ← 1 + (2+5)*5;
293:    21335     A[3,3] ← 3*5;
294:    21351     NUMBINVERT(A,B,3);
295:    21360     FOR I←1 STEP 1 UNTIL 3 DO
296:    21410     FOR J←1 STEP 1 UNTIL 3 DO
297:    21441     PRINT(.I=I^.J=J, A[I,J],B[I,J],U[I,J],V[I,J],
298:    21532               SIMPLIFY( V[I,J], 0 ),
299:    21554               EVAL(X)V[I,J](5)  );
300:    21613     END
```

APPENDIX 1 CONTINUED

```
301:        SN          PAGE
302:                    COMMENT:              DESCRIPTION OF THE OUTPUT
303:
304:                    FIRST LINE      THE PAIR I^J.
305:                    SECOND LINE     THE ARRAY ELEMENT A[I,J]  (NUMERIC INPUT MATRIX)
306:                    THIRD LINE      THE ARRAY ELEMENT B[I,J]  (NUMERIC OUTPUT MATRIX
307:                    FOURTH LINE     THE ARRAY ELEMENT U[I,J]  (SYMBOLIC INPUT MATRIX
308:                    FIFTH LINE      THE ARRAY ELEMENT V[I,J]  (SYMBOLIC OUTPUT MATRI
309:                    SIXTH LINE      THE SIMPLIFICATION OF V[I,J].
310:                    SEVENTH LINE    THE EVALUATION (X=5) OF V[I,J].
311:                    THIS LAST LINE MATCHES THE THIRD LINE;
```

```
BEGIN EXECUTION 23:33:12;  .04500 AVAILABLE CELLS
I=1^J=1
5
.20000000000.+00
X
1/X
1/X
(.20000000000.+00)
I=1^J=2
0
0
0
0
0
0
I=1^J=3
0
0
0
0
0
0
I=2^J=1
1
-.11111111111.-01
1
(0 - 1*1/X)*1/(2*X + X + 3)
-(1/(3*X + 3*X↑2))
(-.11111111111.-01)
```

APPENDIX 1 CONTINUED

```
I=2^J=2
18
.55555555556E-01
2*X + X + 3
1/(2*X + X + 3)
1/(3 + 3*X)
(.55555555556E-01)
I=2^J=3
0
0
0
0
0
0
I=3^J=1
6
-.53333333333E-01
1 + X
(0 - (1 + X)*1/X - (1 + (2 + X)*X)*(n - 1*1/X)*1/(2*X + X + 3))*1/(3*X)
-((2*X + 4*X↑2 + 2*X↑3)/(9*X↑3 + 9*X↑4))
(-.53333333333E-01)
I=3^J=2
36
-.13333333333E+00
1 + (2 + X)*X
(0 - (1 + (2 + X)*X)*1/(2*X + X + 3))*1/(3*X)
-((1 + 2*X + X↑2)/(9*X + 9*X↑2))
(-.13333333333E+00)
I=3^J=3
15
.66666666667E-01
3*X
1/(3*X)
1/(3*X)
(.66666666567E-01)
```

APPENDIX 2

AN ITERATIVE DIFFERENTIATION ALGORITHM

```
001:            BEGIN
002:  11002     FORM X,F,G,H;
003:  11014     REAL ARRAY Y[0:20];
004:  11031     FORM PROCEDURE DIFF(G,X); VALUE G,X; FORM G,X;
005:            BEGIN
006:            COMMENT:   THIS PROCEDURE COMPUTES THE DERIVATIVE OF G
007:            WITH RESPECT TO X USING AN ITERATIVE METHOD.  THE BASIC
008:            STRATEGY OF THE PROCEDURE IS THIS. WE HAVE TWO FORMULA
009:            ARRAYS, F AND D. F[I] CONTAINS A FORMULA TO BE DIFFEREN-
010:            TIATED AND D[I] WILL CONTAIN THE EXPRESSION OF ITS
011:            DERIVATIVE. THIS DERIVATIVE IS CONSTRUCTED, IN GENERAL,
012:            IN TERMS OF THE DERIVATIVES OF THE OPERANDS OF F[I],
013:            THESE OPERANDS WILL BE STORED IN SOME OTHER F[K] AND F[J]
014:            WITH K, J > I. IN THE EXPRESSION OF D[I] WE DO NOT USE THE
015:            VALUES OF D[K] AND D[J] RATHER WE USE THE REPRESENTATION
016:            OF THE ACCESS TO THEIR VALUES (DOT-ARRAY). ULTIMATELY
017:            THE PROCESS OF DIFFERENTIATION REDUCES TO DIFFERENTIATION
018:            OF X AND CONSTANTS, THUS CAUSING IT TO TERMINATE. WHEN THIS
019:            HAPPENS WE COLLECT BACKWARDS BY MEANS OF EVAL;
020:  11051     SYMBOL CUALQ;
021:  11056     INTEGER I,J,K; FORM A; FORM ARRAY F,D[1:20];
022:  11077     SWITCH L ← L1,L2,L3,L4,L5,L6,L7,L8,L9,L10;
023:  11141     SYMBOL T,Q; INTEGER P;
024:  11146     T←/[OPERATOR:+,-,*,/,↑][INDEX:P];
025:  11226     Q←/[OPERATOR:LN,EXP,SIN,COS,-][INDEX:P];
026:  11306     CUALQ←[REAL,INTEGER,HALF,FORM];
027:  11335     F[1] ← G; J←1;
028:  11351     FOR I←1 STEP 1 UNTIL J DO
029:            BEGIN
030:  11402     IF F[I] == ATOM THEN BEGIN
031:  11422     D[I] ← IF F[I] == X THEN 1 ELSE 0; GOTO CONTINUE END;
032:  11457     IF F[I] == Y.[A:INTEGER] THEN BEGIN
033:  11521     K←EVAL A; D[I] ← Y.[K+1]; GOTO CONTINUE END;
034:  11561     IF F[I] == F[J+1]:OF(CUALQ) AND F[J+2]:OF(CUALQ) THEN
035:  11646     GOTO L[P] ELSE
036:  11652     IF F[I] == IQ[ F[J+1]:OF(CUALQ) THEN GOTO L[P+5] ELSE
037:  11722     COMMENT ERROR; PRINT(999999); GOTO EXIT;
038:  11727     L1: D[I] ← D.[J+1] + D.[J+2]; GOTO NEXT;
039:  12005     L2: D[I] ← D.[J+1] - D.[J+2]; GOTO NEXT;
040:  12063     L3: D[I] ← D.[J+1]*F[J+2] + F[J+1]*D.[J+2]; GOTO NEXT;
041:  12171     L4: D[I] ← (F[J+2]*D.[J+1] - F[J+1]*D.[J+2])/F[J+2]↑2;
042:  12320                 GOTO NEXT;
043:  12323     L5: D[I] ← F[J+2]*F[J+1]↑(F[J+2]-1)*D.[J+1]  +
044:  12424                   LN(F[J+1])*F[I]*D.[J+2]; GOTO NEXT;
045:  12506     L6: D[I] ← D.[J+1]/F[J+1]; GOTO NEXT1;
046:  12553     L7: D[I] ← F[I]*D.[J+1]; GOTO NEXT1;
047:  12621     L8: D[I] ← COS(F[J+1])*D.[J+1]; GOTO NEXT1;
048:  12672     L9: D[I] ← -SIN(F[J+1])*D.[J+1]; GOTO NEXT1;
049:  12745     L10:D[I] ← -D.[J+1]; GOTO NEXT1;
050:  13002     NEXT: J←J+1;
051:  13006     NEXT1: J←J+1;
052:            CONTINUE:
053:  13012     PRINT( I, F[I], D[I] );
054:  13033     END OF LOOP;
```

APPENDIX 2 CONTINUED

```
055:  13036    FOR I←J STEP -1 UNTIL 1 DO BEGIN
056:  13067    PRINT( I, D[I] );
057:  13102    D[I] ← EVAL D[I];
058:  13125    PRINT( D[I] );
059:  13135    END;
060:  13140    DIFF←D[1];
061:           EXIT:
062:  13150    END OF PROCEDURE DERV;
063:  13153    F ← X↑2 + (X-1)*(X+1) - SIN(X)/LN(X);
064:  13231    H←DIFF(F,X);
065:  13242    PRINT(H);
066:  13245    END;
```

```
BEGIN EXECUTION 11:33:31;  .06242 AVAILABLE CELLS
1
X↑2 + (X - 1)*(X + 1) - SIN(X)/LN(X)
(D.[2]) - (D.[3])
2
X↑2 + (X - 1)*(X + 1)
(D.[4]) + (D.[5])
3
SIN(X)/LN(X)
(LN(X)*(D.[6]) - SIN(X)*(D.[7]))/LN(X)↑2
4
X↑2
2*X↑(2 - 1)*(D.[8]) + LN(X)*X↑2*(D.[9])
5
(X - 1)*(X + 1)
(D.[10])*(X + 1) + (X - 1)*(D.[11])
6
SIN(X)
COS(X)*(D.[12])
7
LN(X)
(D.[13])/X
8
X
1
9
2
0
10
X - 1
(D.[14]) - (D.[15])
11
X + 1
(D.[16]) + (D.[17])
12
X
1
13
X
1
14
X
1
15
1
0
```

APPENDIX 2 CONTINUED

```
16
X
1
17
1
0
17
0
0
16
1
1
15
0
0
14
1
1
13
1
1
12
1
1
11
(D.[16]) + (D.[17])
1
10
(D.[14]) - (D.[15])
1
9
0
0
8
1
1
7
(D.[13])/X
1/X
6
COS(X)*(D.[12])
COS(X)
5
(D.[10])*(X + 1) + (X - 1)*(D.[11])
X + 1 + X - 1
4
2*X↑(2 - 1)*(D.[8]) + LN(X)*X↑2*(D.[9])
2*X
3
(LN(X)*(D.[6]) - SIN(X)*(D.[7]))/LN(X)↑2
(LN(X)*COS(X) - SIN(X)*1/X)/LN(X)↑2
2
(D.[4]) + (D.[5])
2*X + X + 1 + X - 1
1
(D.[2]) - (D.[3])
2*X + X + 1 + X - 1 - (LN(X)*COS(X) - SIN(X)*1/X)/LN(X)↑2
2*X + X + 1 + X - 1 - (LN(X)*COS(X) - SIN(X)*1/X)/LN(X)↑2
```

APPENDIX 3

AN ITERATIVE ALGORITHM FOR SOLVING AN EQUATION ON X

```
001:            BEGIN
002:  11002     FORM G, K, M, H, N, P, A, B, C, X; SYMBOL OP1,OP2;
003:  11034     INTEGER I; SWITCH L ← L1,L2,L3,L4,L5;
004:  11057     SWITCH Q ← Q1, Q2,Q3,Q4,Q5,Q6, Q7;
005:  11107     OP1 ← /(OPERATOR:*,+,-,/,↑)[(INDEX:I)];
006:  11162     OP2 ← /(OPERATOR:-,EXP,LN,SQRT,ARCTAN,SIN,COS)[(INDEX:I)];
007:  11247     G ← K↑2 + LN( M + SIN((X↑3-K)/(H+4)*M↑5)↑N - K)*M = P;
008:
009:
010:  11354     FOR G ← G WHILE ¬(G == X=ANY) DO BEGIN
011:  11377     IF G == (A:ANY (OP1) B:ANY) = C:ANY THEN GOTO L[I];
012:  11451     IF G == (+(OP2) A:ANY) = C:ANY THEN GOTO Q[I];
013:  11512     PRINT(,NOEQUATION); GOTO EXIT;
014:  11520     L1: G← IF A>>X THEN A=C/B ELSE B=C/A; GOTO CONTINUE;
015:  11555     L2: G←IF A>>X THEN A=C-B ELSE B=C-A; GOTO CONTINUE;
016:  11612     L3: G←IF A>>X THEN A=B+C ELSE B=A-C; GOTO CONTINUE;
017:  11647     L4: G←IF A>>X THEN A=C*B ELSE B=A/C; GOTO CONTINUE;
018:  11704     L5: G←IF A>>X THEN A=C↑(1/B) ELSE B=LN(C)/LN(A);
019:  11752           GOTO CONTINUE;
020:  11755     Q1: G←A=-C; GOTO CONTINUE;
021:  11770     Q2: G←A=LN(C); GOTO CONTINUE;
022:  12003     Q3: G←A=EXP(C); GOTO CONTINUE;
023:  12016     Q4: G←A=C↑2; GOTO CONTINUE;
024:  12034     Q5: G←A=SIN(C)/COS(C); GOTO CONTINUE;
025:  12057     Q6: G←A=ARCTAN(C/SQRT(1-C↑2)); GOTO CONTINUE;
026:  12112     Q7: G←A=ARCTAN(SQRT(1-C↑2)/C); GOTO CONTINUE;
027:  12145     CONTINUE: END;
028:  12147     EXIT: PRINT(G) END;
0  ERRORS
```



```
EGIN EXECUTION 23:21:07;  .06518 AVAILABLE CELLS
=(K + ARCT((K + EXP((P - K↑2)/M) - M)↑(1/N)/SQRT(1 - (K + EXP((P - K↑2)/M) - M)↑(1/N)↑2))/M↑5*(H + 4))↑(1/3)
```

A RECURSIVE ALGORITHM FOR SOLVING AN EQUATION ON X

```
001:  11002    BEGIN FORM E,K,M,N,H,P,F,G,X;
002:  11026    SYMBOL PLUS, TIMES;
003:  11033    BOOLEAN PROCEDURE HASX(F); VALUE F; FORM F; HASX←F>>X;
004:  11053    PLUS ←/[OPERATOR:+][COMM:TRUE];
005:  11103    TIMES ←/[OPERATOR:*][COMM:TRUE];
006:
007:           BEGIN
008:  11133    FORM PROCEDURE SOLVE(LHS,RHS); FORM LHS,RHS;
009:  11140    BEGIN FORM A, B, C;
010:  11151    A←A:OF(HASX); B←B:ANY; C←C:ANY;
011:                  SOLVE ←
012:  11203    IF LHS == (A|PLUS|B) THEN SOLVE( A, RHS-B) ELSE
013:  11246    IF LHS == (A|TIMES|B) THEN SOLVE(A, RHS/B) ELSE
014:  11311    IF LHS == A - B THEN SOLVE( A, RHS+B) ELSE
015:  11352    IF LHS == B - A THEN SOLVE( A, B-RHS) ELSE
016:  11413    IF LHS == A / B THEN SOLVE( A, RHS*B) ELSE
017:  11454    IF LHS == B / A THEN SOLVE( A, B/RHS) ELSE
018:  11515    IF LHS == A ↑ B THEN SOLVE( A, RHS↑(1/B) ) ELSE
019:  11565    IF LHS == B ↑ A THEN SOLVE( A, LN(RHS)/LN(B)) ELSE
020:  11634    IF LHS == -A THEN SOLVE( A, -RHS) ELSE
021:  11671    IF LHS == EXP(A) THEN SOLVE( A, LN(RHS)) ELSE
022:  11726    IF LHS == LN(A) THEN SOLVE( A, EXP(RHS)) ELSE
023:  11763    IF LHS == SQRT(A) THEN SOLVE( A, RHS↑2) ELSE
024:  12023    IF LHS == ARCTAN(A) THEN SOLVE(A, SIN(RHS)/COS(RHS)) ELSE
025:  12072    IF LHS == SIN(A) THEN SOLVE(A, ARCTAN(RHS/SQRT(1-RHS↑2))) ELSE
026:  12153    IF LHS == COS(A) THEN SOLVE(A, ARCTAN(SQRT(1-RHS↑2)/RHS)) ELSE
027:  12234    IF LHS == X THEN X=RHS ELSE .NOEQUATION;
028:  12263    END;
029:
030:  12266    FORM PROCEDURE ANSWER(E); FORM E;
031:  12271    BEGIN FORM F,G;
032:  12300    IF E == G:ANY=F:ANY THEN BEGIN IF F>>X THEN
033:  12345    ANSWER←SOLVE(F,G) ELSE ANSWER←SOLVE(G,F) END ELSE
034:  12370    ANSWER←.NOEQUATION; END;
035:  12376    E ← K↑2 + LN(M + SIN((X↑3-K)/(H+4)*M↑5)↑N -K)*M = P;
036:  12503    PRINT(E,ANSWER(E));
037:  12515    END END
```

```
BEGIN EXECUTION 16:44:44;  .06408 AVAILABLE CELLS
K↑2 + LN(M + SIN((X↑3 - K)/(H + 4)*M↑5)↑N - K)*M=P
X=(ARCT((EXP((P - K↑2)/M) + K - M)↑(1/N)/SQRT(1 - (EXP((P - K↑2)/M) + K - M)↑(1/N)↑2))/M↑5*(H + 4) + K)↑(1/3
```

APPENDIX 5

A MARKOV ALGORITHM FOR SOLVING AN EQUATION ON X

```
001:            BEGIN
002:  11002     FORM E,K,M,H,N,P;
003:  11020     FORM A,B,C,X; SYMBOL PLUS, TIMES, S;
004:  11037     BOOLEAN PROCEDURE HASX(F); VALUE F; FORM F;
005:  11051     HASX ← F >> X;
006:  11060     A←A:OF(HASX);
007:  11072     B←B:ANY;
008:  11103     C←C:ANY;
009:  11114     PLUS←/[OPERATOR:+][COMM:TRUE];
010:  11144     TIMES←/[OPERATOR:*][COMM:TRUE];
011:  11174     S ← [
012:            [
013:  11177     (A:TIMES:B) = C    →    .A = .C / .B,
014:  11243     (A:PLUS :B) = C    →    .A = .C - .B,
015:  11310     A     -     B = C    →    .A = .C + .B,
016:  11353     B     -     A = C    →    .A = .B - .C,
017:  11416     A     /     B = C    →    .A = .C * .B,
018:  11461     B     /     A = C    →    .A = .B / .C,
019:  11524     A     ↑     B = C    →    .A = .C ↑ (1/.B),
020:  11574     B     ↑     A = C    →    .A = LN(.C)/LN(.B),
021:  11643           -     A = C    →    .A = -.C,
022:  11677         EXP(A)    = C    →    .A = LN(.C),
023:  11733          LN(A)    = C    →    .A = EXP(.C),
024:  11767        SQRT(A)    = C    →    .A = .C ↑ 2,
025:  12026      ARCTAN(A)    = C    →    .A = SIN(.C)/COS(.C),
026:  12073         SIN(A)    = C    →    .A = ARCTAN(.C/SQRT(1-.C↑2)),
027:  12152         COS(A)    = C    →    .A = ARCTAN(SQRT(1-.C↑2)/.C),
028:  12231              X    = C    →    .X = .C ] ];
029:  12263     E ← K↑2 + LN(M + SIN( (X↑3-K)/(H+4)*M↑5 )↑N - K)*M  =  P;
030:  12370     PRINT( E, E.↓S );
031:  12400     END;
```

PAGE 169

```
BEGIN EXECUTION 17:12:57;  .06424 AVAILABLE CELLS
K↑2 + LN(M + SIN((X↑3 - K)/(H + 4)*M↑5)↑N - K)*M=P
X=(ARCT((EXP((P - K↑2)/M) + K - M)↑(1/N)/SQRT(1 - (EXP((P - K↑2)/M) + K - M)↑(1/N)↑2))/M↑5*(H + 4) + K)↑(
.33333333333₁₀+00)
```

APPENDIX 6

A MARKOV ALGORITHM FOR TRANSFORMING LOGICAL FORMULAE INTO DISJUNCTIVE NORMAL FORM

```
001:  11002    BEGIN FORM A, B, C, R, X;
002:  11016    SYMBOL S,Y1;
003:
004:  11023    Y1←/[OPERATOR:∧][COMM:TRUE];
005:  11053    A←A:ANY; B←B:ANY; C←C:ANY;
006:
007:  11104          S ← [[
008:  11107          ¬ (A ∧ B)    →   ¬.A ∨ ¬.B,
009:  11144          ¬ (A ∨ B)    →   ¬.A ∧ ¬.B,
010:  11202          ¬ A ¬= B     →   .A  = .B,
011:  11234          ¬ A  = B     →   .A ¬= .B,
012:  11266          ¬ A ¬< B     →   .A  < .B,
013:  11320          ¬ A  < B     →   .A ¬< .B,
014:  11352          ¬ A ¬> B     →   .A  > .B,
015:  11404          ¬ A  > B     →   .A ¬> .B
016:  11417          ],
017:  11440          (A∨B) |Y1| C  →  .A∧.C ∨ .B∧.C ];
018:
019:  11517    R ← ¬(¬(X<8∧X>-10)∧X=5)∧¬¬(X¬<15);
020:  11576    PRINT( R, R.←S);
021:  11606    END
```

```
BEGIN EXECUTION 18:36:03;  »06630 AVAILABLE CELLS
¬(¬(X<8∧X>-10)∧X=5)∧¬¬X¬<15
X<8∧X>-10∧X<15 ∨ X≠5∧X<15
```

APPENDIX 7

A PROGRAM TO SOLVE A DIFFERENTIAL EQUATION BY COMPUTING THE FIRST N TERMS OF A TAYLOR SERIES

```
087:          SN        PAGE
088:   14234      FORM PROCEDURE SOLVEDIFF(DE,X,K,N,A);
089:   14237      VALUE DE,X,K,N,A; FORM DE,X; REAL A; INTEGER N,K;
090:             COMMENT:
091:             THIS PROCEDURE SOLVES A DIFFERENTIAL EQUATION 'DE'.
092:             Y.[0] IS THE UNKNOWN FUNCTION, Y.[I] IS THE I-TH DERIVATIVE OF Y.[0]
093:             RESPECT TO X. DE IS A FUNCTION OF X, Y.[0], .., Y.[K].
094:             FURTHERMORE DE CONTAINS ONLY ONE OCCURRENCE OF Y.[K].
095:             Y[0], .... Y[K] CONTAIN THE VALUES OF Y.[0], .., Y.[K] FOR X=A.
096:             THIS PROCEDURE COMPUTES THE FIRST N TERMS OF THE EXPANSION OF Y.[0]
097:             IN THE NEIGHBORHOOD X=A;
098:   14270     BEGIN FORM G,RH; INTEGER I;
099:   14277     G ← SOLVE( DE, Y.[K] );
100:   14330     IF G == Y.[K] = RH:ANY THEN BEGIN
101:   14367     FOR I←K+1 STEP 1 UNTIL N DO BEGIN
102:   14421     RH ← DIFF(RH,X);
103:   14433     Y[I] ← EVAL(X)RH(A) END;
104:   14476     G ← Y[0];
105:   14506     FOR I←1 STEP 1 UNTIL N DO
106:   14537     G ← G + (Y[I]/FACTORIAL(I))*(X-A)↑I;
107:   14611     SOLVEDIFF ← G END ELSE
108:   14615     PRINT(.ERROR);
109:   14621     END;
```

APPENDIX 8

A MARKOV ALGORITHM TO ELIMINATE UNARY MINUS

```
001:  11002    BEGIN FORM A, B, C, N;
002:  11014    SYMBOL MINUSOUT, P, T;
003:
004:
005:  11023    A ← A:ANY;  B ← B:ANY;  C ← C:ANY;  N ← N:ANY;
006:  11064    P←/[OPERATOR:+][COMM:TRUE];  T←/[OPERATOR:*][COMM:TRUE];
007:
008:
009:  11143                          MINUSOUT←[[
010:  11146    -(-A)                 .→   .A                          ,
011:  11166    -A  +  -B             .→   -(.A+.B)                    ,
012:  11226    -A  -  -B             .→   .B-.A                       ,
013:  11264    (-A)*(-B)             .→   .A*.B                       ,
014:  11322    (-A)/(-B)             .→   .A/.B                       ,
015:  11360    -(A-B)                .→   .B - .A                     ,
016:  11412    -((A-B)/C)            .→   (.B-.A)/.C                  ,
017:  11457    -(C/(B-A))            .→   .C/(.A-.B)                  ,
018:  11524    -A |P| B              .→   .B-.A                       ,
019:  11562     A  -  -B             .→   .A+.B                       ,
020:  11614    -A  -   B             .→   -(.A+.B)                    ,
021:  11650    (-A) |T| B             →   -(.A*.B)                    ,
022:  11710     (-A)/B                →   -(.A/.B)                    ,
023:  11744    A/(-B)                .→   -(.A/.B)                    ,
024:  12000    (-A)↑N                .→   (-1)↑.N * .A↑.N             ,
025:  12046    A↑(-B)                .→   1/.A↑.B                     ,
026:  12105    .XXX→.ZZZ]];
027:
028:  12125    END
```

APPENDIX 9

A MARKOV ALGORITHM TO CLEAR RATIONAL FRACTIONS

```
001:  11002    BEGIN FORM A, B, C, D, N;
002:  11016    SYMBOL P, T, RATIONAL;
003:
004:  11025    A ← A:ANY;  B ← B:ANY;  C ← C:ANY;  D ← D:ANY;  N ← N:ANY;
005:  11076    T ← /[OPERATOR:*][COMM:TRUE];
006:  11126    P ← /[OPERATOR:+][COMM:TRUE];
007:
008:  11156                        RATIONAL←[[
009:  11161    (A/B)*(C/D)          .→   (.A*.C)/(.B*.D)          ,
010:  11240    (A/B)/(C/D)          .→   (.A*.D)/(.B*.C)          ,
011:  11320     (A/B) ITI C          →   (.A*.C)/.B               ,
012:  11367    (A/B)/C              .→   .A/(.B*.C)               ,
013:  11432    C/(A/B)              .→   (.B*.C)/.A               ,
014:  11475    (A/B)↑N              .→   .A↑.N / .B↑.N            ,
015:  11547    .XXX→.ZZZ],[
016:  11566    A/B + C/D            .→   (.A*.D + .C*.B)/(.B*.D)  ,
017:  11663    A/B - C/D            .→   (.A*.D - .C*.B)/(.B*.D)  ,
018:  11761    A/B IPI C            .→   (.A + .C*.B)/.B          ,
019:  12037    A/B  -  C            .→   (.A - .C*.B)/.B          ,
020:  12111    C - A/B              .→   (.C*.B - .A)/.B          ,
021:  12163    .XXX→.ZZZ]];
022:
023:  12204    END
```

APPENDIX 10

A MARKOV ALGORITHM TO EXPAND A FORMULA BY MULTIPLYING OUT FACTORS

```
0001:  11000
0002:  11020    BEGIN FORM A, B, C, D, N, N1;
0003:  11040    SYMBOL P, T, EXPAND;
0004:
0005:  11070    A ← A:ANY;  B ← B:ANY;  C ← C:ANY;
0006:  11110    T ← /[OPERATOR:*][COMM:TRUE];
0007:  11140    P ← /[OPERATOR:+][COMM:TRUE];
0008:  11170    T ← /[OPERATOR:*][COMM:TRUE];
0009:  11200
0010:  11203                          EXPAND←[[
0011:  11233    ((A+B) !T! C) !T! D    →   (.A+.B)*(.C*.D)      ,
0012:  11320    ((A-B) !T! C) !T! D    →   (.A-.B)*(.C*.D)      ,
0013:  11406    (A*B)↑N                .→  .A↑.N * .B↑.N        ,
0014:  11450    .XXX→.ZZZ],[
0015:  11467    (A+B)↑(2)              .→  .A↑2 + .B↑2 + 2*.A*.B ,
0016:  11560    (A-B)↑(2)              .→  .A↑2 + .B↑2 - 2*.A*.B ,
0017:  11652    (A+B)↑N                .→  (.A+.B)↑(.N-1)*(.A+.B) ,
0018:  11740    (A-B)↑N                .→  (.A-.B)↑(.N-1)*(.A-.B) ,
0019:  11806    (A↑N1)↑N               .→  .A↑(.N1*.N)          ,
0020:  12071    (A+B) !T! C            .→  .A*.C + .B*.C        ,
0021:  12147    (A-B) !T! C            .→  .A*.C - .B*.C        ,
0022:  12205    A !P! (B-C)             →  (.A+.B)-.C           ,
0023:  12272    A  -  (B-C)            .→  (.A+.C)-.B           ,
0024:  12335    (B-C)  -  A            .→  .B - (.C+.A)         ,
0025:  12400    .XXX→.ZZZ]];
0026:
0027:  12421    END
```

APPENDIX 11

A PROCEDURE TO COLLECT ALIKE TERMS IN A POLYNOMIAL AND AUXILIARY PROCEDURES

```
117:          SN        PAGE              PROCEDURE: COLLECT
118:    13454 AL PROCEDURE COLLECT(F); VALUE F; FORM F;
119:    13464    BEGIN FORM A; REAL C1,C2,C3,C4; INTEGER J1,J2; UU←.B;
120:    13473    I←J1←I+1; IF F==P15 THEN BEGIN
121:    13506    A←AX; UU←.A-.B; TERMS(BX); C←3; JU←.B-.C;
122:    13543    FOR K←J1 STEP 1 UNTIL I DO E[K]←-E[K] END
123:    13615    ELSE A←F; TERMS(A); J2←I; C1←0; I1←J1;
124:    13633    FOR K←J2 STEP -1 UNTIL J1+1 DO
125:    13664    IF ¬(M[K]==0) THEN BEGIN
126:    13704    EXTRACT(K); M[K]←AX;
127:    13722    IF ABS(KX)=1 THEN BEGIN
128:    13726    C1←C1+KX*(EVAL BX); M[K]←0 END ELSE
129:    13761    BEGIN A←BX; C3←CX*SIGN(KX); C4←C3; C2←NX;
130:    13776    FOR J←K-1 STEP -1 UNTIL J1+1 DO
131:    14030    IF ¬(M[J]==0) THEN BEGIN
132:    14050    IF ABS(E[J])=ABS(E[K]) THEN BEGIN
133:    14071    EXTRACT(J); IF C2=NX THEN BEGIN
134:    14101    IF A==BX THEN BEGIN
135:    14110    C4←C4 + CX*SIGN(KX); M[J]←0 END END END END;
136:    14134    IF C4 ¬= C3 THEN BEGIN
137:    14140    IF C4=0 THEN M[K]←0 ELSE BEGIN
138:    14156    M[K]←IF ABS(C4)=1 THEN A ELSE ABS(C4)*A; E[K]←ABS(E[K])*SIGN(C4)
139:    14220    END END END END; I1←J1; COMPACT;
140:    14236    M[I1]←ABS(C1); E[I1]←SIGN(C1); ARRANGE;
141:    14270    END;
```

```
142:          SN        LINE 3            PROCEDURE: MAKEF
143:    14272           FORM PROCEDURE MAKEF;
144:    14274           BEGIN FORM G1,G2;
145:    14302           FOR K←I1 STEP 1 UNTIL I-1 DO
146:    14333    IF E[K]=E[K+1] THEN BEGIN INTEGER K1,K2;
147:    14356    K2←I;
148:    14360    FOR K1←K STEP 1 UNTIL K2-1 DO
149:    14411    FOR J←K STEP 1 UNTIL K2-K1 DO
150:    14443    IF E[J]=E[J+1] THEN BEGIN INTEGER J1,J2,J3;
151:    14466    J1←I+1; BREAK(M[J],1,P12,P11);
152:    14511    J2←I; BREAK(M[J+1],1,P12,P11);
153:    14534    FOR J3←J1 STEP 1 UNTIL J2 DO
154:    14564    IF E[J3]<E[J3+J2-J1+1] THEN BEGIN
155:    14610    AX←M[J]; M[J]←M[J+1]; M[J+1]←AX; J3←J2+1 END;
156:    14655    I←J1-1 END ELSE K2←J; K←J END;
157:    14675    IF I=I1 THEN BEGIN
158:    14700    AX ← IF E[I1]=0 THEN 0 ELSE
159:    14712         IF E[I1]<0 THEN -M[I1] ELSE M[I1]; GOTO POLY1 END;
160:    14745    G1← M[I1]; FOR J←I1+1 STEP 1 WHILE J<I+1∧E[J]>0 DO
161:    15022    G1←M[J]+G1;
162:    15040    IF E[J]=0 THEN J←J+1;
163:    15053    IF J>I THEN BEGIN
164:    15057    AX←G1; GOTO POLY1 END;
165:    15063    G2←M[I];
166:    15072    FOR K←I-1 STEP -1 UNTIL J DO G2←M[K] + G2;
167:    15141    IF J<3 THEN BEGIN
168:    15145    IF E[I]<0 THEN BEGIN
```

APPENDIX 11 CONTINUED

```
169:  15156     AX← -(M[I]+G2); GOTO POLY1 END END;
170:  15175     AX ← G1-G2;
171:  15203     POLY1: MAKEF←AX END;


172:          SN        LINE 3          PROCEDURE: POLY
173:  15207 AL FORM PROCEDURE POLY(F); VALUE F; FORM F;
174:  15217     BEGIN COLLECT(F); POLY←MAKEF; I←I1-1 END;


037:          SN        LINE 2          PROCEDURE: ORDER
038:  11265     INTEGER PROCEDURE ORDER(F); FORM F;
039:  11267     IF F==OF(LEXICO) THEN
040:  11303     ORDER ← ILX TH OF THE ORD OF .LEXICO ELSE ORDER←0;


041:          SN        LINE 2          PROCEDURE: BREAK
042:  11336     PROCEDURE BREAK(F,S,P1,P2); VALUE F,S;
043:  11340     FORM F,P1,P2; INTEGER S;
044:  11352     BEGIN NX←1; IF F==P1 THEN BEGIN
045:  11365     I←I+1; M[I]←AX; E[I]←NX*S END ELSE
046:  11414     IF F==P2 THEN BEGIN FORM B; B←BX; BREAK(AX,S,P1,P2);
047:  11443     BREAK(B,S,P1,P2) END ELSE PRINT(P1^P2^F) END;


048:          SN        LINE 2          PROCEDURE: TESTIDEN
049:  11476 AL PROCEDURE TESTIDEN(A,B); INTEGER A,B;
050:  11500     BEGIN INTEGER C1,C2; C1←C2←1;
051:  11505     FOR K←I STEP -1 UNTIL I1+1 DO
052:  11536     IF ¬(M[K]==0) THEN BEGIN
053:  11556     IF M[K] == OF(NUM) THEN BEGIN
054:  11575     IF E[K]>0 THEN
055:  11606     C1←C1*(EVAL M[K]) ELSE
056:  11627     C2←C2*(EVAL M[K]); M[K]←0 END ELSE
057:  11662     FOR J←K-1 STEP -1 UNTIL I1+1 DO
058:  11714     IF M[J] == M[K] THEN BEGIN
059:  11740     M[J]←0; E[K]←E[K]+E[J] END END;
060:  12004     A←C1; B←C2 END;


061:          SN        LINE 2          PROCEDURE: ARRANGE
062:  12016     PROCEDURE ARRANGE;
063:  12020     FOR K←I1 STEP 1 UNTIL I-1 DO BEGIN
064:  12051     FOR J←I1 STEP 1 UNTIL I-1+I1-K DO BEGIN
065:  12105     IF E[J]<E[J+1] THEN BEGIN
066:  12127     NX←E[J]; AX←M[J];
067:  12145     E[J]←E[J+1]; M[J]←M[J+1];
068:  12203     E[J+1]←NX+0;M[J+1]←AX; END END END;


069:          SN        LINE 2          PROCEDURE: COMPACT
070:  12235     PROCEDURE COMPACT; BEGIN
071:  12237     Q1: IF I>I1 THEN BEGIN
072:  12243     IF M[I]==0 THEN BEGIN I←I-1; GOTO Q1 END;
073:  12266     FOR K←I-1 STEP -1 UNTIL I1+1 DO
074:  12320     IF M[K]==0 THEN BEGIN
075:  12340     M[K]←M[I]; E[K]←E[I]; I←I-1 END END;
076:  12401     END;
```

```
77:          SN        PAGE

78:          SN        LINE 2          PROCEDURE EXTRACT
79:   12402     PROCEDURE EXTRACT(I); VALUE I; INTEGER I;
80:   12410     IF M[I]==P17 THEN BEGIN KX←E[I];
81:   12435     IF ¬(AX==P18) THEN BEGIN
82:   12443     CX←1; BX←AX END END ELSE PRINT(.ERR^M[I]);


83:          SN        LINE 3          PROCEDURE TERMS
84:   12467     PROCEDURE TERMS(F); FORM F;
85:   12471     BEGIN INTEGER J1,J2,J3; FORM G; REAL C1,C2,C3,C4;
86:   12475     J1←I+1; BREAK(F,1,P16,P14); J2←I;
87:   12512     FOR J3←J1 STEP 1 UNTIL J2 DO BEGIN C1←C2←C3←C4←1;
88:   12550     G←M[J3]; I←I1←J2←I;
89:   12563     IF G==ATOM THEN BEGIN C4←ORDER(G); GOTO TERM1 END;
90:   12603     IF G==P10 THEN BEGIN C4←ORDER(AX); C1←C2←C3←NX; GOTO TERM1 END;
91:   12627     BREAK(G,1,P12,P11);
92:   12637     TESTIDEN(C1,C2);
93:   12645     M[I1]←C1; E[I1]←1;
94:   12670     COMPACT; C1←C2←E[I1+1]; C3←0; C4←1;
95:   12707     FOR K←I1+1 STEP 1 UNTIL I DO BEGIN
96:   12740     J←E[K];  IF J>C1 THEN C1←J;
97:   12756     C3←C3+J; IF J<C2 THEN C2←J;
98:   12767     NX←ORDER(M[K]); E[K]←NX;
99:   13017     M[K]← IF J¬=1 THEN M[K]↑J ELSE M[K];
100:  13055     C4←C4*NX END; C3← ENTIER(C3/(I-I1));
101:  13070     ARRANGE;
102:  13072     G←M[I1]; IF I>I1 THEN BEGIN
103:  13105     FOR J←I1+1 STEP 1 UNTIL I-1 DO
104:  13137     G←M[J]*G;
105:  13155     IF ¬(M[I]==1) THEN G←M[I]*G END ELSE C1←C2←C3←1;
106:  13214     TERM1: C1←C1-C2; C3←C3-C2; C2←C2-1;
107:  13225     BX ← IF C1*(.6931471487) + C3*(1.098612257) + C2*(1.609437881) > 29.
108:  13237     THEN  11 ELSE 2↑C1 * 3↑C3 * 5↑C2;
109:  13266     M[J3]←G^BX; E[J3]←C4;
110:  13314     IF PASO1>0 THEN BEGIN
111:  13320     IF PASO2=0 THEN BEGIN B←IF M[J1]==P17 THEN AX ELSE M[J1];
112:  13352     FOR J←J1+1 STEP 1 UNTIL J2 DO B←B+(IF M[J]==P17 THEN AX ELSE M[J])
113:  13432     ; YY←REPLACE(UU); CONPRINT END; END;
114:  13446     END;
115:  13450     I←J2;
   N
116:  13452     END PROCEDURE TERM;
```

APPENDIX 12

PROCEDURE FOR FACTORING OUT CONSTANTS AND VARIABLES

AS WELL AS FOR COMPLETING SQUARES

```
185:          SN       LINE 3          PROCEDURE FACTOR
186:  15565     BOOLEAN PROCEDURE FACTOR(F,H); VALUE F; FORM F,H;
187:            COMMENT:  FACTOR IS TRUE IF THE FORMULA F CAN BE FACTOR
188:            AND/OR PUT AS A SUM OF SQUERES, THE VALUE OF THE FORMULA IS
189:            STORED IN H.  F MUST BE A SUM OF TERMS AS PROCESSED
190:            BY EXPAND;
191:  15576     BEGIN INTEGER J,J1,J2,J3,K,K1,K2,K3,K4,K5,K6;
192:  15601     BOOLEAN BOOL; FORM A,B,G; SYMBOL Y;
193:  15612     COLLECT(F); COMPACT; J1←I1; J2←I; J3←J2-J1+1; G←1;
194:  15635     FOR K←J1 STEP 1 UNTIL J2 DO E[K+J3]←ABS(E[K]);
195:  15711     K1← GCD(J2+1,J2+J3,K2);
196:  15733     IF K1=1 THEN GOTO L1; BOOL←TRJE;
197:  15743     FOR K4←J2+J3+1 STEP 1 UNTIL K2 DO BEGIN
198:  15776     AX ← E[K4]; IF ¬(AX==OF(PRIM)) THEN GOTO ERR;
199:  16021     AX←(IP+1) TH OF LEXICO; Y←[AX+CX:INTEGER,AX]; Y←/[INDEX:K3];
200:  16110     FOR J←J1 STEP 1 UNTIL J2 DO BEGIN
201:  16141     BX←.YY; IF N1:M[J] >> BX:OF(Y) THEN BEGIN
202:  16201     M[J+J3]←N1; E[J+J3]←IF K3=2 THEN 1 ELSE CX END ELSE GOTO ERR END;
203:  16240     K3←E[J2+1]; FOR K←J2+2 STEP 1 UNTIL J2+J3 DO IF K3>E[K] THEN K3←E[K];
204:  16326     G←G*AX↑K3; FOR K←J1 STEP 1 UNTIL J2 DO BEGIN
205:  16372     K1←E[K+J3]-K3; BX←1;
206:  16410     BX ← IF K1=1 THEN AX ELSE (
207:  16416           IF K1=0 THEN BX ELSE AX↑(1);
208:  16433     M[K]←EVAL(.YY)M[K+J3](BX) END END;
209:  16511     L1: FOR K←J1 STEP 1 UNTIL J2 DO BEGIN
210:  16542     IF M[K]==P18 THEN BEGIN
211:  16562     M[K+J3]←BX; E[K+J3]←CX END ELSE
212:  16610     IF M[K] == OF(NUM) THEN BEGIN
213:  16630     M[K+J3]←1; E[K+J3]←EVAL M[K] END ELSE GOTO L2 END;
214:  16674     K1← GCD(J2+1,J2+J3,K2);
215:  16716     IF K=1 THEN GOTO L2; G←K1*G; BOOL←TRUE; FOR K←J1 STEP 1 UNTIL J2 DO
216:  16765     M[K] ← IF E[K+J3]=1 THEN M[K+J3] ELSE E[K+J3]*M[K+J3];
217:  17046     L2: I←J2; I1←I+1; K3←J2;
218:  17056     FOR K←J1 STEP 1 UNTIL J2 DO BEGIN
219:  17107     A←M[K]; IF A==P18 THEN BEGIN
220:  17127     K1←SQRT(CX) END ELSE BEGIN
221:  17134     BX←A; K1←1 END;
222:  17141     I←J2+J3; I1←I+1; BREAK(BX,1,P12,P11); A←1;
223:  17163     FOR J←I1 STEP 1 UNTIL I DO BEGIN
224:  17214     K2←E[J]/2; IF E[J]¬=2*K2 THEN GOTO SIGUE;
225:  17244     A←A*M[J]↑K2 END;
226:  17267     K3←K3+1; M[K3]←K1*A; E[K3]←K;
227:  17322     SIGUE: END;
228:  17325     IF K3<J2+2 THEN GOTO L3; K4←PASO1; PASO1←0; B←0;
229:  17344     FOR K←J2+1 STEP 1 UNTIL K3-1 DO BEGIN K5←E[K];
230:  17407     FOR J← K+1 STEP 1 UNTIL K3 DO BEGIN K6←E[J];
231:  17451     IF E[K5]*E[K6] => 0 THEN GOTO C2;
232:  17474     I←K3; I1←I+1; TERMS(2*M[K]*M[J]);
233:  17537     IF ¬(M[K3+1] == P17) THEN GOTO ERR;
234:  17561     FOR K1←J1 STEP 1 UNTIL J2 DO BEGIN
235:  17612     IF K1=K5 ∨ K1=K6 THEN GOTO C1;
```

APPENDIX 12 CONTINUED

```
236:  17633     IF E[K1]=0 THEN GOTO C1;
237:  17646     IF M[K1]=-AX THEN BEGIN
238:  17665     A ← ( IF E[K5]*E[K1]>0 THEN(M[K]+M[J]) ELSE (M[K]-M[J]))↑2;
239:  17760     B ← IF E[K5]>0 THEN (B+A) ELSE (B-A);
240:  20006     E[K1]←0; E[K5]←0; E[K6]←0;
241:  20042     BOOL←TRUE; GOTO C2 END; C1: END; C2: END END;
242:  20055     L3: FACTOR←BOOL; IF BOOL THEN BEGIN
243:  20063     FOR K←J1 STEP 1 UNTIL J2 DO
244:  20114     IF E[K]¬=0 THEN
245:  20127     B ← B + (IF E[K]>0 THEN M[K] ELSE -M[K]);
246:  20167     IF ¬(B==0) THEN G←G*B;
247:  20206     H← EVAL G END; GOTO L4;
248:  20222     ERR:PRINT(.ERR←F); L4: PASO1←<4;
249:  20234     I1←J1; I←I1-1; END;

250:          SN       LINE 2
251:
252:
253:  20244     PRIM ← [2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53];
254:  20407     PRIM←/[INDEX:IP];
```

```
      SN       LINE 2          PROCEDURE: PRIME
14542 AL  INTEGER PROCEDURE PRIME(I); VALUE I; INTEGER I;
14551     BEGIN INTEGER N,J,K,K1,K2,K3; SYMBOL J1;
14556     N ← COUNT(PRIM); IF N<I THEN BEGIN
14570     FOR J←N+1 STEP 1 UNTIL I DO BEGIN
14622     K←LAST OF PRIM;
14646     FOR K1←K+2 STEP 2 WHILE TRUE DO BEGIN
14675     FOR J1← ELEMENTS OF PRIM DO BEGIN
14722     K2←J1; K3←K1/K2; IF K3*K2=K1 THEN GOTO L1; IF K2*2>K1 THEN GOTO L3 END;
14755     L3: J1←K1; INSERT J1 AFTER LAST OF PRIM; GOTO L2;
15007     L1: END; L2:END;
15014     PRIME←K1 END ELSE PRIME← I TH OF PRIM;

15042     END PROCEDURE PRIME;
```

APPENDIX 12 CONTINUED

```
      SN        LINE 2          PROCEDURE GCD
15045     INTEGER PROCEDURE GCD(I1,I2,I3); VALUE I1,I2; INTEGER I1,I2,I3;
          COMMENT:  GCD IS THE GREATER COMMON DIVISOR FOR THE I INTEGERS:
        E[I1] TO E[I2]. E[I2+1] TO E[I3] WILL HAVE THE PRIMES OF GCD;
15060     BEGIN INTEGER C1,C2,C3,C4,J,J1,K;
15063     C1←1; C2←E[I1];
15075     FOR J←I1 STEP 1 UNTIL I2 DO BEGIN
15126     IF E[J]=1 THEN GOTO DONE;
15142     IF E[J]<C2 THEN C2←E[J] END;
15165     K←I2; C3←1; FOR J←1 STEP 1 WHILE C2>C3 DO BEGIN
15221     C3←PRIME(J);
15232     L1: C4←C2/C3;
15236     IF C2=C4*C3 THEN BEGIN
15243     K←K+1; E[K]←C3;
15260     IF C4=1 THEN GOTO L2;
15266     C2←C4; GOTO L1 END END;
15274     PRINT( (C2=.ERR)=C3 );
15312     L2: I3←I2; FOR J1←I2+1 STEP 1 UNTIL K DO BEGIN C3←E[J1];
15360     FOR J←I1 STEP 1 UNTIL I2 DO BEGIN
15411     C2←E[J]; C4←C2/C3;
15425     IF C2¬=C4*C3 THEN GOTO NEXT;
15435     E[K+J]←C4 END;
15452     C1←C1*C3; I3←I3+1; E[I3]←C3;
15502     FOR J←I1 STEP 1 UNTIL I2 DO E[J]←E[J+K];
15555     NEXT: END;

15560     DONE: GCD←C1; END PROCEDURE GCD;

21714     F ← (3*A↑4*B*A*B + A↑2*3*B*2*A*B↑2*A + C*B*D*6*A*B*A↑2 +
22065          2*A*5*A*B*3*B*A + 12*B↑5*A↑5 + 3*A*B*B↑3*A↑2 ) -
22232         (3*B↑2*A↑3*C↑2+3*B↑2*A↑3*D*D);
22344     PRINT( F, FACTOR(F,F), F);
22360          END
```

```
BEGIN EXECUTION 15:55:17;   04353 AVAILABLE CELLS:
3*A↑4*B*A*B + A↑2*3*B*2*A*B↑2*A + C*B*D*6*A*B*A↑2 + 2*A*5*A*B*3*B*A + 12*B↑5*A↑5 + 3*A*B*B↑3*A↑2 -
(3*B↑2*A↑3*C↑2 + 3*B↑2*A↑3*D*D)
TRUE
3*A↑3*B↑2*((A + B)↑2 - (D - C)↑2 + 10 + 4*A↑2*B↑3)
```

APPENDIX 13

TWO MARKOV SHEMATA FOR GENERATING EQUIVALENT RELATIONS IN INEQUALITY PROOFS BY LOGICAL CHAIN

```
372:          SN       LINE 2          PROCEDURE POSITIVE
373:   22207 AL BOOLEAN PROCEDURE POSITIVE(F); VALUE F; FORM F;
374:   22220    POSITIVE←TESTSIGN(F)=1;

375:          SN       LINE 2          PROCEDURE NEGATIVE
376:   22234 AL BOOLEAN PROCEDURE NEGATIVE(F); VALUE F; FORM F;
377:   22245    NEGATIVE←TESTSIGN(F)=-1;
```

```
419:   23137    A←A:ANY; B←B:ANY; C←C:ANY; N←N:INTEGER; N1←N1:INTEGER;
420:   23207    D←D:ANY;
421:   23217    P←/[OPERATOR:+][COMM:TRUE]; T←/[OPERATOR:*][COMM:TRUE];
422:   23275    POS←POS:OF(POSITIVE); NEG←NEG:OF(NEGATIVE);
423:   23317    DIV ← /[OPERATOR:/][COMM:TRUE];
```

```
476:          SN       LINE 2
477:   26477                        IFF1 ← [[
478:   26501    POS < SQRT(A)          .→   .POS↑2 < .A          ,
479:   26536    SQRT(A) < B            .→   .A < .B↑2            ,
480:   26574    A/POS < C              .→   .A < .POS*.C         ,
481:   26636    A/NEG < C              .→   .NEG*.C < .A         ,
482:   26700    C < A/POS              .→   .C*.POS < .A         ,
483:   26742    C < A/NEG              .→   .A < .NEG*.C         ,
484:   27004    .XXX→.ZZZ]];

485:          SN       LINE 2
486:   27023                        IFF2 ← [[
487:   27025    0 < (A IDIVI POS)      .→   0 < .A               ,
488:   27060    0 < (A IDIVI NEG)      .→   .A < 0               ,
489:   27114    (A IDIVI POS) < (0)    .→   .A < 0               ,
490:   27150    (A IDIVI NEG) < (0)    .→   0 < .A               ,
491:   27204    0 < -A                 .→   .A < 0               ,
492:   27234    -A < (0)               .→   0 < .A               ,
493:   27264    0 < (POS ITI A)        .→   0 < .A               ,
494:   27320    0 < (NEG ITI A)        .→   .A < 0               ,
495:   27354    (POS ITI A) < (0)      .→   .A < 0               ,
496:   27410    (NEG ITI A) < (0)      .→   0 < .A               ,
497:   27444    .XXX→.ZZZ]];
```

PROCEDURES RELATED TO LIMIT PROCESSES

```
    SN      PAGE          PROCEDURE: LIMIT


    SN      LINE 3
15767       FORM PROCEDURE LIMIT(F,X,T,DIREC); VALUE F,X,T,DIREC;
            COMMENT THIS PROCEDURE COMPUTES:

            LIMIT F(X)  WHEN DIREC = 0    OR    LIMIT F(X)  WHEN DIREC = +1
             X→T                                X→T[+]

            LIMIT F(X)  WHEN DIREC = -1
            X→T[-]
                    IT WILL OUTPUT AS VALUE OF LIMIT ANY ONE OF THE FOLLOWING:
            1.- A NON ZERO NUMBER.
            2.- A FORMULA NOT CONTAINING X.
            3.- <INFI,ZERO> | -<INFI,ZERO> | SIGN(<FORM WITHOUT X>)*<INFI,ZERO>
            4.- THE NUMBER '0'.

            ZERO = LIMIT X → 0[+], THE NUMBER 0 IS USED WHEN LIMIT F(X) = 0 AND
                   X→0[+]                                                X→A
            FOR EVERY EPSILON>0 THERE ARE X1 AND X2 SUCH THAT |X1-A|<EPSILON
            AND |X2-A|<EPSILON AND F(X1)>0 AND F(X2)<0;

 5772       FORM F, X,T; INTEGER DIREC;
16021       BEGIN FORM V; INTEGER D;
            BOOLEAN HOPE;


    SN      LINE 3          PROCEDURE: AUXLIM


    SN      LINE 3
16026       FORM PROCEDURE AUXLIM(F,CARE); VALUE F,CARE;
16032       FORM F; BOOLEAN CARE;
 6045       BEGIN FORM LIMA, LIMB, A, B; INTEGER J1;
16060       SWITCH Q ← Q1,Q2,Q3,Q4,Q5,Q6,Q7,Q8,Q9,Q10,HOPELESS;

    SN      LINE 1          PROCEDURE: LHOSPITAL
 6125       FORM PROCEDURE LHOSPITAL(A,B); VALUE A,B; FORM A,B;
16144       BEGIN FORM AY, BY, CY, DA, DB; INTEGER VECES;
 6161       VECES←4; AY←A; BY←B;
 6170       ITER: CY ← SIMPLIFY(DERV(SIMPLIFY(AY),X)/DERV(SIMPLIFY(BY),X) );
16237       IF ¬(CY == AY:ANY/BY:ANY) THEN BEGIN AY←CY; BY←1 END;
16276       DA ← AUXLIM(AY,CARE); DB ← AUXLIM(BY,CARE);
 6317       CY ← LI(4,DA,DB,CARE);
16332       IF CY==TRUE THEN BEGIN
16343       IF I¬=2 THEN GOTO LHOSERR;
 6352       IF  VECES>0 THEN BEGIN VECES←VECES-1; IF VECES=0 THEN
 6365       BEGIN VECES←-4; AY←1/A; BY←1/B END; GOTO ITER END;
16407       IF VECES<0 THEN BEGIN VECES←VECES+1; GOTO ITER END;
            COMMENT: PONER AQUI EL ANALISIS ALTERNATIVO CUANDO REGLA NO FUNCIONA
 6420       LHOSERR: CY←UNDEF; PRINT(,ERRHOSP←A←B) END;
```

```
16436           LHOSPITAL ← CY END;

      SN        LINE 1          PROCEDURE LIMOPERAND
16443           PROCEDURE LIMOPERAND(F,LIMF,CARE,AHEAD);
16446           VALUE F, CARE; FORM F, LIMF; BOOLEAN CARE; LABEL AHEAD;
                COMMENT: THIS PROCEDURE COMPUTES THE LIM OF OPERAND F
                USING HEURISTIC I;
16461           BEGIN FORM A;
16466           A ← AUXLIM(F, FALSE);
16477           IF ¬HOPE THEN GOTO HOPE_ESS;
16506           IF ¬CARE THEN GOTO IF A==UNDEF THEN Q1 ELSE L1;
16526           IF (A==UNDEF)∨(A==P3) THEN BEGIN
16544           A ← AUXLIM(F, TRUE);
16555           IF ¬HOPE THEN GOTO HOPE_ESS;
16564           IF A==UNDEF THEN GOTO TRYHARDER END;
16576           L1: LIMF ← A; GOTO AHEAD;

16605           END PROCEDURE LIMOPERAND;

      SN        LINE 1          BEGINS PROCEDURE AUXLIM
16610           IF ¬(F>>X) THEN BEGIN AUXLIM ← F; GOTO FIN END;
16625           IF   F==X  THEN BEGIN AUXLIM ← Y; GOTO FIN END;
16643           IF F==P1 THEN BEGIN J1←J+5; A←A1;
                                                        ↓↓↓
16660           CC1: LIMOPERAND( A, LIMA, CARE, CC2) END;
16671           IF F == P2 THEN BEGIN
                                                                  ↓↓↓
16701           J1 ← J; A← A1; B← A2; LIMOPERAND( B, LIMB, CARE, CC1) END;
16720           GOTO HOPELESS;

      SN        LINE 1
16723           CC2: CX ← LI(J1, LIMA, LIMB, CARE);
16736           IF CX == TRUE THEN GOTO Q(I);
16750           AUXLIM ← CX; GOTO FIN;



      SN        LINE 3          PROBLEMATIC CASES



      SN        LINE 3
                TRYHARDER:
                HOPELESS:
16755           Q0: HOPE ← FALSE;
16761           Q1: AUXLIM ← UNDEF; GOTO FIN;
                Q2: COMMENT ZERO/ZERO OR INFI/INFI OR ZERO*INFI;
16766           IF J1=4 THEN BEGIN AUXLIM ← LHOSPITAL(A,B); GOTO FIN END;
17004           AUXLIM ← LHOSPITAL(A, 1/B); GOTO FIN;
                Q3: COMMENT 1^INFI;
                Q4: COMMENT 0^0 OR INFI^0;
17027           AUXLIM ← LI^12, LHOSPITAL( LN(A),1/B ), ANY, CARE); GOTO FIN;
                Q5: COMMENT A^B WITH A<0;
17075           IF ¬ENTERO(B) THEN GOTO Q0;
17110           BOOL ← LIMB==OF(QB);
```

```
17117           AX ← IF KB=0 THEN (IF JB=0 THEN 1 ELSE .BOUNDED) ELSE (-1)↑LIMB;
17142           A ← EVAL AX; IF LIMA == -1 THEN BEGIN AUXLIM←A; GOTO FIN END;
17165           A ← A*LI(5,-LIMA,LIMB,CARE);
17211           AUXLIM ← EVAL A; GOTO FIN;
                Q6: COMMENT INFI← INFI;
17222           BEGIN FORM D;
17227           D ← LHOSPITAL(B,A); IF ¬(D==1) THEN BEGIN
17250           AUXLIM ← LI(3, LIMA, LI(1,1,D,CARE),CARE); GOTO FIN END;
17300           AUXLIM ← LHOSPITAL( A, 1/(1+B/A)); GOTO FIN END;
                Q7: COMMENT A↑(<UNKNOWN SIGN>*INFI);
                Q8: COMMENT <LN, SQRT>(<UNKNOWN SIGN>*INFI);
                Q9: COMMENT 0↑0, 0*0, INFI*INFI WITH CARE;
                Q10: COMMENT COMPUTATION WITH NUMBERS YIELDS ZERO;
                FIN:

17335           END PROCEDURE AUXLIM;


     SN         LINE 2          PROCEDURE RLIM


     SN         LINE 3
17340           FORM PROCEDURE RLIM(K); VALUE K; INTEGER K;
                COMMENT THIS PROCEDURE COMPUTES THE RESTRICTED LIMIT.
                K IS THE DIRECTION;
17347           BEGIN INTEGER C, I, J; BOOLEAN M; D←K; C←I←J←1; M←TRUE;
17363           IF T==0 THEN BEGIN C←K; I←-1; J←0; GOTO AHI END;
17404           IF T == INFIP THEN GOTO AHI;
17416           IF T == INFIM THEN BEGIN C←-1;
17430           AHI: Y←ZEROINFI.(C,I,0,J) END ELSE
17471           IF Y == .Y THEN Y ← SIMPLIFY(T);
17512           HOPE ← TRUE;
17516           CX ← AUXLIM(F, FALSE);
17527           BBB: IF ¬HOPE THEN BEGIN RLIM←UNDEF; GOTO EXITR END;
17540           BOOL ← CX==DF(DA); IF KA>2 THEN BEGIN
17552           RLIM ← CX; GOTO EXITR END;
17557           IF (KA=1)∨(FA==2) THEN BEGIN
17575           IF M THEN BEGIN CX←AUXLIM(F,TRUE); M←FALSE;GOTO BBB END;
17614           IF KA=1 THEN JA←1;
17622           RLIM ← IF JA=0 THEN .ZERO ELSE UNDEF; GOTO EXITR END;
17635           AX ← SIMPLIFY(CA); I←TESTSIGN(AX);
17654           CX ← IF JA= 0 THEN .ZEROP ELSE .INFIP;
17666           RLIM ← IF I=0 THEN SIGN(AX)*CX ELSE
17702                  (IF I=1 THEN CX ELSE
17710                  (IF JA=0 THEN .ZEROM ELSE .INFIM));

17722           EXITR: END PROCEDURE RLIM;


     SN         LINE 2          BEGINS BODY PROCEDURE LIMIT

     SN         LINE 1
17725           AX←1;
17731 SN        CMPL 1770006324
```

```
17733 SN        CMPL 4150000015
17735 SN        CMPL 1330042300
17737           PRINT(.TIME=TIME^.CELL=CELLS);
17764 SN        CMPL 1770005316
17766 SN        CMPL 1770005316
17770           PRINT(.F=F);
20000 SN        CMPL 1770005316
20002 SN        CMPL 1770005316

      SN        LINE 2
20004           IF T==INFIP THEN BEGIN AX←RLIM(-1); GOTO FUERA END;
20030           IF T==INFIM THEN BEGIN AX←RLIM( 1); GOTO FUERA END;
20051           IF DIREC ¬= 0 THEN BEGIN AX←RLIM(DIREC); GOTO FUERA END;
20067           BEGIN FORM LEFT, RIGHT;
20076           LEFT ← RLIM(1);
20105           IF LEFT == UNDEF THEN BEGIN AX←UNDEF; GOTO FUERA END;
20122           RIGHT ← RLIM(-1);
20135           IF LEFT == RIGHT THEN BEGIN AX←LEFT; GOTO FUERA END;
20151           IF (LEFT>>OF(CERO))∧(RIGHT>>OF(CERO)) THEN
20171           BEGIN AX←.ZERO; GOTO FUERA END;
20177           AX ← UNDEF;
20202           PRINT(LEFT:LEFT∧RIGHT:RIGHT);
20227           END;
20231           FUERA: LIMIT ← AX;
20234 SN        CMPL 1770005316
20236 SN        CMPL 4150000015
20240 SN        CMPL 1330042300
20242 SN        CMPL 1770005316
20244           PRINT( LIMIT.(.F) = AX, X←T);
20277 SN        CMPL 1770005316
20301           PRINT(.DIREC=DIREC);
20312 SN        CMPL 1770005316
20314 SN        CMPL 1770005316
20316           PRINT(.TIME=TIME^.CELL=CELLS);

20343           END PROCEDURE LIMIT;
```

```
SN        LINE 3          THE EVALUATOR


SN        LINE 3
12666     FORM PROCEDURE LI(J1,A,B,CARE); VALUE J1,A,B,CARE;
12671     FORM A,B; INTEGER J1; BOOLEAN CARE;
          COMMENT:

           THIS PROCEDURE ACOMPLISHES THE LIMIT EVALUATION, J1 IS THE OPERATOR
           INDEX (RELATIVE TO ARRAY BASICFORM). A AND B ARE OPERANDS;

          BEGIN
12716     SWITCH O ← O1,O2,O3,O4,O5,O6,O7,O8,O9,O10,O11,O12;
12770     SWITCH S ← S1 , S3 , S2 , S5 , S4 ,
13003                S10, S7 , S6 , S9 , S8 ;
13032     PROCEDURE BNOTZERO; PRINT(ASSUME:B¬=0);
13054     PROCEDURE ANOTZERO; PRINT(ASSUME:A¬=0);
13076     PROCEDURE ANOTINFI; PRINT(ASSUME:A¬=INFI);
13117     PROCEDURE BNOTINFI; PRINT(ASSUME:B¬=INFI);
13140     PROCEDURE ANOTB;    PRINT(ASSUME:A¬=-B);
13166     PROCEDURE AXGRZERO; PRINT(ASSUME:AX>0);

13207     IF CELLS<400 THEN GC;
13215     BOOL ← A == OF (QA); IF KA=1 THEN GOTO UNDEFCASE;
13231     AX ← IF KA=2 THEN CA ELSE A;
13241     JJA ← TESTSIGN(AX);
13251     IF J1<6 THEN BEGIN BOOL ← B == OF(QB);

          COMMENT WHEN BINARY OPERATOR (J1<6) WE HAVE THE FOLLOWING SITUATION

                    KB  1          2          3          4
          KA  A       B  UNDEF      ZEROINFI   NUMBER¬=0  FORMULA
          1   UNDEF      SPECIAL    SPECIAL    SPECIAL    SPECIAL
          2   ZEROINFI   SPECIAL      I=1        I=2        I=3
          3   NUMBER¬=0  SPECIAL      I=4      NOPROBLEM  NOPROBLEM
          4   FORMULA    SPECIAL      I=5      NOPROBLEM  NOPROBLEM

          KA=2 → A IS ZEROINFI.(CA,IA,FA,JA)  JA=0 → ZERO  JA=1 → INFI
          KB=2 → B IS ZEROINFI.(CB,IB,FB,JB)  JB=0 → ZERO  JB=1 → INFI

          WHEN KA=2 AND KB=2, I.E. WHEN I=1 THEN VALUES OF KA AND KB ARE CHANGE
          AS FOLLOWS, SAY KX IS ANY ONE OF THEM
                         ZEROINFI IS
          KX=0  WHEN     CX*Z↑IX
          KX=1  WHEN     SIGN IS THAT OF CX, BUT BEHAIVOR IS UNKNOWN
          KX=2  WHEN     BOTH SIGN AND BEHAIVOR ARE UNKNOWN
          KX=3  WHEN     FX(Z)                                        ;
```

```
13264           IF KB=1 THEN GOTO UNDEFCASE;
13272           AX ← IF KB=2 THEN CB ELSE B;
13302           JJB ← TESTSIGN(AX);
13312           IF J1=5 THEN BEGIN
13316           IF KA=3 THEN BEGIN
13322           IF NA<0 THEN GOTO OSA;
13331           IF KB=4^NA=1 THEN BEGIN BNOTINFI; GOTO S11A END END END;
13353           IF KA¬=2^KB¬=2 THEN GOTO NOPROBLEM;
13373           I←IF KA=2 THEN KB-1 ELSE KA+1;
13405           IF I=1 THEN BEGIN
13411           IF ¬(FA==KA:INTEGER) THEN KA←3;
13430           IF ¬(FB==KB:INTEGER) THEN KB←3 END;

13447           GOTO O(J1) END OF BINARY CASE, NOW FOLLOWS UNARY;
13452           IF (J1=7 ∨ J1=11) THEN BEGIN
13470           IF JJA<0 THEN GOTO HOPELESS;
13477           IF JJA=0 THEN AXGRZERO;
13505           IF A==P3 THEN BEGIN I←8; GOTO SIPROBLEM END END;
13521           IF KA¬=2 THEN GOTO NOPROBLEM; GOTO O(J1);

      SN        PAGE            ROUTINES FOR EACH OPERATOR


      SN        LINE 3
13532           01:  GOTO S(I1); COMMENT     '+'

                              ZEROINFI  NUMBER¬=0  FORMULA
                ZEROINFI        S1          S3        S2
                NUMBER¬=0       S5       ¬CONSIDER ¬CONSIDER
                FORMULA         S4       ¬CONSIDER ¬CONSIDER;

                02:    COMMENT    '-';

13535           LI ← LI(1,A,LI(6,B,ANY,CARE),CARE); GOTO EXIT;

13570           03:  GOTO S(I+5); COMMENT '*'

                              ZEROINFI  NUMBER¬=0  FORMULA
                ZEROINFI        S10         S7        S6
                NUMBER¬=0       S9       ¬CONSIDER ¬CONSIDER
                FORMULA         S8       ¬CONSIDER ¬CONSIDER;

                04:      COMMENT '/';
13574           B ← SIMPLIFY(1/B);
13614           IF I=1 THEN GOTO OX3;
13622           IF KB=2 THEN BEGIN
13626           OX3: IF FB==FORM THEN FB←1/FB;
13644           CB←1/CB; CB←EVAL CB; JB←INVERT(JB);
13670           IB ← 0-IB END; GOTO 03;
```

```
O5:   COMMENT '↑'

        |-1 IF A IS NEGATIVE
JJA =   | 0 IF A IS A FORM WITH UNKNOWN SIGN
        | 1 IF A IS POSITIVE

JJB IS COMPUTED SIMILARLY FROM B

                ZEROINFI  NUMBER¬=0  FORMULA
ZEROINFI          I=1       S12        S12
NUMBER¬=0         S11     ¬CONSIDER  ¬CONSIDER
FORMULA           S11     ¬CONSIDER  ¬CONSIDER;

13675   JJB ← TESTSIGN(IF KB=2∨I=1 THEN CB ELSE B);
13727   IF JJA<0 THEN BEGIN
13734   O5A:  I←5; GOTO SIPROBLEM END;
13741   IF JJA=0 THEN AXGRZERO;
13747   IF I=1 THEN BEGIN
13753   IF JB=0 THEN BEGIN I←4; GOTO SIPROBLEM END;
13763   NA ← JA*100; GOTO S11B END;
13771   IF KA=2 THEN GOTO S12; GOTO S11;

14001   O6: COMMENT 'UNARY -'; CA←-CA; CA←EVAL CA; GOTO S10A;

        O7: COMMENT 'SQRT';
14015   B ← 0.5; NB←0+1/2; JJB←1; KB←0+3; GOTO S12;

        O8:  COMMENT 'ABS';
14034   IF FA ==2 THEN BEGIN FA←1; CA←1 END;
14052   CA←ABS(CA); GOTO S10A;

14061   O9: COMMENT 'SIN'; IF JA=1 THEN BEGIN LI←.BOUNDED; GOTO EXIT END;
14072   LI ← ZEROINFI(JJA,0,IF JJA=0 THEN 2 ELSE 1, 0 ); GOTO EXIT;

14117   O10: COMMENT 'COS'; IF JA=0 THEN GOTO S11A; LI ← .BOUNDED;GOTO EXIT;

        O11: COMMENT 'LN';

14132   IF FA==0 THEN BEGIN CA←IA; FA←LN(Z) END ELSE
14152   BEGIN FA←1; CA←IF JA=0 THEN -1 ELSE 1 END;
14166   JA←1; GOTO S10A;

        O12: COMMENT 'EXP';

14173   IF JA=0 THEN GOTO S11A;
14201   IF FA==2 THEN GOTO S11C;
14213   IF JJA=0 THEN BEGIN PRINT(ASSUME:CA¬=0);
14234   LI ← .IF CA>0 THEN INFI ELSE 0; GOTO EXIT END;
14263   JA ← IF JJA=1 THEN 1 ELSE 0; CA←FA←1; GOTO S10A;
```

SN PAGE LAST PART OF LIMIT EVALUATION PROCEDURE 'LI'

SN LINE 3

```
14302          S1: IF JA¬=JB THEN GOTO S3;
14311          IF KA=2 ∨ KB=2 THEN BEGIN
14327          S1A: IF JA=1̄ THEN BEGIN I←6; GOTO SIPROBLEM END;
14337          S1B: IF CARE THEN BEGIN I←9; GOTO SIPROBLEM END;
14347          LI ← ZEROINFI.(1,1,2,JA); GOTO EXIT END;
14410          IF KA=1 ∨ KB=1 THEN BEGIN
14426          IF 1 = TESTSIGN(SIMPLIFY(CA*CB)) THEN BEGIN
14456          S1F: FA←1; GOTO S10A END; GOTO S1A END;
14465          IF KA=0∧KB=0̄ THEN BEGIN
14503          IF IA>IB THEN GOTO FIRSTOPR;
14512          IF IB>IA THEN GOTO SECONOPR;
14521          S1D: AX ← SIMPLIFY(CA+CB); IF AX==0 THEN GOTO S1A;
14550          IF AX==FORM THEN PRINT(ASSUME:AX¬=0); CA←AX; GOTO S10A;
14600          END; GOTO S1A;
14602          S2: IF JA=1 THEN ANOTB;
14610          S3: IF JA=1 THEN GOTO FIRSTOPR; GOTO SECONOPR;
14620          S4: IF JB=1 THEN ANOTB;
14626          S5: JA←INVERT(JB); GOTO S3;
14637          S6: IF JA=0 THEN BNOTINFI ELSE BNOTZERO;
14650          S7: CA←CA*B; CA←EVAL CA; GOTO S10A;
14666          S8: IF JB=0 THEN ANOTINFI ELSE ANOTZERO;
14677          S9: CB ← CB*A; CB ← EVAL CB;
14713          LI ← ZEROINFI.(CB,IB,FB,JB); GOTO EXIT;
14754          S10: CA←CA*CB; CA←EVAL CA;
14770          IF JA ¬= JB THEN BEGIN
14775          IF KA=0∧KB=0̄ THEN BEGIN
15013          IA ← IA+IB;
15017          IF IA=0 THEN BEGIN LI←CA; GOTO EXIT END;
15027          IF IA<0 THEN JA←0 ELSE JA←1;
15041          GOTO S10A END;
15044          S10B: I←2 ; GOTO SIPROBLEM END;
15051          IF KA=2 ∨ KB=2 THEN GOTO S1B;
15070          IF KA=1∨KB=1̄ THEN GOTO S1F;
15107          S10C: IF KA=0∧KB=0 THEN IA←IA+IB ELSE
15131          BEGIN IF KA=0 THEN FA←Z↑IA; IF KB=0 THEN FB←Z↑IB; FA←FA*FB END;
15164          S10A: LI ← ZEROINFI.(CA,IA,FA,JA); GOTO EXIT;
15225          S11: IF JB=0̄ THEN BEGIN
15231          IF KA=4 THEN BEGIN ANOTZERO; ANOTINFI END;
15241          S11A: LI ← 1; GOTO EXIT END;
15247          S11B: IF FB == 2 THEN BEGIN
15257          S11C: I←7; GOTO EXIT END;
15264          IF KA=4 THEN BEGIN
15270          AX ← .IF A<1̄ THEN 0 ELSE INFI;
15315          LI ← IF JJB>0 THEN AX   ELSE
15324                IF JJB<0 THEN 1/AX ELSE
15340              .IF  CB>0 THEN AX   ELSE 1/AX;
15371          GOTO EXIT END;
15374          IF NA=1 THEN BEGIN I←3;GOTO SIPROBLEM END;
15404          CA←FA←IA←1;
15413          JA ← IF NA<1̄ THEN 0 ELSE 1;
```

```
15424          IF JJB<0 THEN JA←INVERT(JA); GOTO S10A;
15441          S12: IF JJB=0 THEN BEGIN BNOTZERO;
15447          AX ← .IF B>0 THEN 0 ELSE INFI;
15474          LI ← IF JA=0 THEN AX ELSE 1/AX; GOTO EXIT END;
15513          CA←CA↑B;
15522          CA ← EVAL CA;
15530          IF KB=3 THEN IA←IA*NB;
15537          IF FA == FORM THEN FA←FA↑B;
15554          IF JJB<0 THEN JA←INVERT(JA); GOTO S10A;
15571          FIRSTOPR: LI ← A; GOTO EXIT;
15576          SECONOPR: LI ← B; GOTO EXIT;
15603          HOPELESS: I←11; GOTO SIPROBLEM;
15610          UNDEFCASE: I←1;
15613          SIPROBLEM: LI ← TRUE; GOTO EXIT;
15621          NOPROBLEM: AX←SIMPLIFY( EVAL(.A,.B)BASICFORM[J1](A,B) );
15703          IF AX == 0 THEN BEGIN
15713          ZEROPROBL: IF CARE THEN BEGIN I←10; GOTO SIPROBLEM END;
15722          AX ← ZEROINFI.(1,-1,2,0) END ;
15761          LI ← AX;

15764          EXIT: END PROCEDURE LI;


1443           FORM PROCEDURE ZEROINFI(C,I,F,J); VALUE C,I,F,J;
11447          FORM C,F; REAL I; INTEGER J;
       BEGIN
               COMMENT:
                        J=0 WHEN ZEROINFI ACTS LIKE ZERO
                        J=1 WHEN ZEROINFI ACTS LIKE INFINITE
               F=0 IF ZEROINFY IS ASYMPTOTICLY EQUIVALENT TO C*Z↑I.
               F=1 IF BEHAIVOR OF ZEROINFI IS UNKNOWN, BUT ITS SIGN IS THAT OF C
               F=2 IF BOTH BEHAIVOR AND SIGN OF ZEROINFI ARE UNKNOWN
               F=FORM IF ZEROINFI IS ASYMPTOTICLY EQUIVALENT TO C*F(Z) AND
                      F IS POSITIVE, I.E. THE SIGN IS THAT OF C;
11474          ZEROINFI ← ZEROINFI.(C,I,F,J);

1552           FORM PROCEDURE SIMPLIFY(F); VALUE F; FORM F;
               BEGIN
11564          G ← F.↓MINUSOUT; G ← G.↓RATIONAL;
1577           SIMPLIFY ← G END;


    SN         LINE 2
1603           INTEGER PROCEDURE INVERT(J);VALUE J;INTEGER J;
               COMMENT:          J=0  → INVERT=1
                                 J=1  → INVERT=0;
1612           INVERT ← ABS(J-1);
               .575:  21302          BASICFORM[1] ← .A+.B;
                576:  21323          BASICFORM[2] ← .A-.B;
                577:  21344          BASICFORM[3] ← .A*.B;
                578:  21365          BASICFORM[4] ← .A/.B;
                579:  21406          BASICFORM[5] ← .A↑.B;
                580:  21427          BASICFORM[6] ← -.A;
                581:  21443          BASICFORM[7] ← SQRT(.A);
                582:  21457          BASICFORM[8] ← ABS(.A);
                583:  21473          BASICFORM[9] ← SIN(.A);
                584:  21507          BASICFORM[10]← COS(.A);
                585:  21523          BASICFORM[11]← LN(.A);
                586:  21537          BASICFORM[12]← EXP(.A);
```

A PROCEDURE TO TEST FOR INTEGER VALUED FUNCTIONS

```
12434           BOOLEAN PROCEDURE ENTERO(F); VALUE F; FORM F;

     SN         LINE 3
                COMMENT:  ENTERO = TRUE IMPLIES F IS AN INTEGER QUANTITY.  THUS
                          ENTERO = FALSE WHEN EITHER F IS NOT AN INTEGER QUANTITY OR
                          THE NATURE OF F COULD NOT BE STABLISHED.
                          THE LIST 'ENTFUN' IS A LIST OF INTEGER VALUED FUNCTIONS;
                BEGIN
12445           IF F == INTEGER THEN BEGIN
12455           VER: ENTERO ← TRUE; GOTO EXITE END;
12462           IF F==OF(ENTFUN) THEN GOTO VER;
12474           IF F==P1 THEN BEGIN
12504           IF J¬=1∧J¬=2 THEN GOTO FALSO;
12524           ENTERO ← ENTERO(A1); GOTO EXITE END;
12536           IF F==P2 THEN BEGIN FORM AX,BX;
12554           IF J=4 THEN GOTO FALSO; AX←A1;BX←A2;
12566           IF J=5∧TESTSIGN(BX)¬=1 THEN GOTO FALSO;
12613           ENTERO ← ENTERO(AX)∧ENTERO(BX); GOTO EXITE END;
12636           FALSO: ENTERO ← FALSE;

12641           EXITE: END PROCEDURE ENTERO;
```

APPENDIX 16

A PROCEDURE TO DETERMINE THE SIGN OF A FORMULA

```
11663           INTEGER PROCEDURE TESTSIGN(F); VALUE F; FORM F;


      SN        LINE 3
                COMMENT:
                TESTSIGN = 1 IF F>0 FOR ALL POSSIBLE VALUES OF PARAMETERS IN F
                TESTSIGN =-1 IF F<0 FOR ALL POSSIBLE VALUES OF PARAMETERS IN F
                TESTSIGN = 0 IF CAN'T TELL OR FOR SURE THERE ARE VALUES OF PARAMETERS
                             IN F FOR WHICH F<0 ANF FOR OTHERS F-<0;
11674           BEGIN REAL R; INTEGER C; FORM CA; SYMBOL QA;
11703           IF F == R:OF(NUM) THEN BEGIN
11722           C ← SIGN(R); GOTO EXITT END;
11733           IF F>0 == OF(GREAT) THEN BEGIN C← 1; GOTO EXITT END;
11757           IF F<0 == OF(GREAT) THEN BEGIN C←-1; GOTO EXITT END;
12003           IF F == P1 THEN BEGIN
12013               C    ← IF J=2∨J=3∨J=7 THEN 1 ELSE
12041                     IF  J=4 ∨ J=5  THEN 0 ELSE
12061                     IF J=6 THEN TESTSIGN(A1-1) ELSE
12104                     -TESTSIGN(A1);              GOTO EXITT END;
12117           IF F == P2 THEN BEGIN INTEGER A,B; FORM AX,BX; C←J;
12137           AX←A1; BX←A2; A←TESTSIGN(AX); B←TESTSIGN(BX);
12162           IF C=1 THEN BEGIN
12166           TS1: IF A=B THEN BEGIN C←A; GOTO TS2 END;
12176           C ← 0; GOTO TS2 END;
12203           IF C=2 THEN BEGIN B←-B; GOTO TS1 END;
12212           IF C=3 ∨ C=4 THEN BEGIN C←A*B;
12233           IF A=B∧C=0 THEN C←IF AX==BX THEN 1 ELSE 0; GOTO TS2 END;
12266           IF A=1 THEN BEGIN C←1; GOTO TS2 END;
12276           IF BX==R:INTEGER THEN BEGIN
12314           IF ABS(R) = ENTIER(ABS(R)/2)*2 THEN BEGIN
12330           C←1; GOTO TS2 END;
12335           C←A END; TS2: END ELSE C←0;
12344           IF C=0 THEN GOTO EXITT;
12352           CA←IF C=1 THEN F>0 ELSE F<0;
12374           QA←CA; INSERT QA AFTER LAST OF GREAT;
12423           PRINT(CA);
                EXITT:
12426           TESTSIGN←C;

12431           END PROCEDURE TESTSIGN;
```

APPENDIX 17

A PROCEDURE FOR ACCOMPLISHING ASYMPTOTIC ANALYSIS

```
215:          SN      PAGE          PROCEDURE REPL
216:  16040     PROCEDURE REPL( F, X,  R );
217:  16043     VALUE F,X; FORM F, X; FORM ARRAY R;
218:            COMMENT:
219:            THIS PROCEDURE CONSTRUCTS, FOR THE INPUT FORMULA F, AN ASYMPTOTICLY
220:            EQUIVALENT FORMULA Q. THE OUTPUT Q IS GIVEN IN A DECOMPOSED FORM:
221:
222:                                    Q = R[3] IF R[3] ¬= 0
223:                                    Q = R[1]*X+R[2] IF R[3] = 0.
224:
225:            THE ALGORITHM IS BASED ON TABLE'S III (CHAPTER IV) AND ON
226:            THE REPLACEMENT THEOREM (APPENDIX A1);
227:            BEGIN
228:  16060     FORM AX, BX, CX;
229:            INTEGER I; REAL EA, EB;
230:  16071     FORM ARRAY RA, RB[1:3];
231:  16110     SWITCH CASE ← MAS, MENOS, POR, ENTRE, ALA, MINUS, RAIZ, ABSO, SEN,
232:  16134                   COSEN, LOG;
233:  16156     A←A;
235:  16161     R[3]←0;
236:  16176     IF ¬(F>>X) THEN BEGIN
237:  16205     R[1]←F; R[2]←0; GOTO EXREP END;
238:  16240     IF F == X THEN BEGIN
239:  16250     R[1]←1; R[2]←1; GOTO EXREP END;
240:  16303     IF F == P2 THEN BEGIN COMMENT F IS BINARY;
241:  16313     AX ← A1; BX←A2; I←J: IF J=5 THEN GOTO ALA;
242:  16327     REPL( AX, X, RA); EA←EVAL RA[2];
243:  16352     L1: REPL( BX, X, RB); EB←EVAL RB[2];
244:  16375     GOTO CASE[I] END;
245:  16400     IF F == P1 THEN BEGIN IF J=7 THEN GOTO EXPO;
246:  16415     I←J+5; BX←A1; GOTO L1 END;
247:  16424     PRINT( .ERR^F); GOTO EXREP;
248:
249:            MAS:
250:  16436     IF RA[3] == 0 THEN BEGIN
251:  16456     IF RB[3] == 0 THEN BEGIN
252:  16476     IF EA > EB THEN BEGIN
253:  16503     R[1]←RA[1]; R[2]←EA; GOTO EXREP END;
254:  16542     IF EA < EB THEN BEGIN
255:  16547     R[1]←RB[1]; R[2]←EB; GOTO EXREP END;
256:  16606     CX ← SIMPLIFY( RA[1] + RB[1], 3);
257:  16642     IF ¬(CX=0) THEN BEGIN
258:  16653     R[1] ←CX; R[2]←EA; GOTO EXREP END;
259:  16705     GO;
260:  16707     CX ← SIMPLIFY( AX+BX, 3);
261:  16727     REPL( CX, X, R); GOTO EXREP END;
262:  16741     RA[3] ← RA[1]*X↑EA;
263:  16773     L4: IF CHECKD( RB[3]/RA[3],X) THEN BEGIN
264:  17031     L3: R[3] ← RA[3] + RB[3]; GOTO EXREP END;
265:  17067     GOTO NOPE END;
266:  17072     IF RB[3] == 0 THEN RB[3]←RB[1]*X↑EB; GOTO L4;
267:
268:            MENOS:

269:  17144     BX←-BX; RB[1]←-RB[1]; RB[3]←-RB[3]; RB[3]←EVAL RB[3]; GOTO MAS;
```

APPENDIX 17 CONTINUED

```
271:              POR:
272:  17234       IF RA[3] == 0 THEN BEGIN
273:  17254       IF RB[3] == 0 THEN BEGIN
274:  17274       R[1] ← RA[1]*RB[1];
275:  17330       R[2] ← EA+EB; GOTO EXREP END;
276:  17350       RA[3] ← RA[1]*X↑EA;
277:  17402       L5: R[3] ← RA[3]*RB[3]; GOTO EXREP END;
278:  17440       IF RB[3] == 0 THEN RB[3]←RB[1]*X↑EB; GOTO L5;
279:
280:              ENTRE:
281:  17512       IF RA[3] == 0 THEN BEGIN
282:  17532       IF RB[3] == 0 THEN BEGIN
283:  17552       R[1] ← RA[1]/RB[1];
284:  17606       R[2] ← EA-EB; GOTO EXREP END;
285:  17626       RA[3] ← RA[1]*X↑EA;
286:  17660       L6: R[3] ← RA[3]/RB[3]; GOTO EXREP END;
287:  17716       IF RB[3] == 0 THEN RB[3] ← RB[1]*X↑EB; GOTO L6;
288:
289:  17770       ALA:  IF B==OF(NUM) THEN BEGIN
290:  20000       REPL( AX, X, RA);
291:  20010       IF RA[3] == 0 THEN BEGIN
292:  20030       R[1] ← RA[1]↑BX;
293:  20055       AX←RA[2]*BX; R[2]←EVAL AX; GOTO EXREP END;
294:  20112       R[3]←RA[3]*BX; GOTO EXREP END;
295:  20141       GOTO NOPE;
296:
297:              MINUS:
298:  20144       R[1] ← -RB[1];
299:  20167       R[2] ←  EB;
300:  20204       A←-RB[3]; R[3]←EVAL A; GOTO EXREP;
301:
302:              RAIZ:
303:  20237       IF RB[3] == 0 THEN BEGIN
304:  20257       R[1] ← SQRT( RB[1] );
305:  20302       R[2] ← EB/2; GOTO EXREP END;
306:  20322       R[3] ← SQRT( RB[3] ); GOTO EXREP;
307:
308:              ABSO:
309:  20347       IF RB[3] == 0 THEN BEGIN
310:  20367       R[1] ← ABS( RB[1] );
311:  20412       R[2] ← EB; GOTO EXREP END;
312:  20431       R[3] ← ABS( RB[3] ); GOTO EXREP;
313:
314:              EXPO:
315:              NOPE:
316:  20456       R[3] ← F;
317:
318:              EXREP:
319:  20472       A ← R[1];
320:  20504       R[1] ← EVAL A;
321:  20524       A ← R[1]; B←R[2]; C←R[3];
322:  20560       D ← IF C==0 THEN A*X↑B ELSE C;
323:  20604       PRINT( ORD.(F)=D );
324:  20630    END PROCEDURE REPLACE;
```

APPENDIX 18

A PROCEDURE AND MARKOV ALGORITHMS FOR CHECKING BOUNDEDNESS

```
104:  11620          FORM PROCEDURE BOUNDP(F); VALUE F; FORM F;
105:                 COMMENT: PROCESSES A FORMULA F FOR BOUNDNESS APPROACH;
106:  11631          BEGIN FORM G;
107:  11636          G ← F.↓Q1;
108:  11644          IF ¬(IBOUND=0) THEN G←G.↓Q2;
      N
109:  11656          BOUNDP ← G END PROCEDURE BOUNDP;
```

```
568:  21100          BOUND ← [ SIN(ANY), COS(ANY) ];
569:  21120          Q1 ← [OF(BOUND) → .BOUNDED];
570:  21137          Q1 ← /[INDEX:!BOUND];
571:  21153          Q2 ← [[BOUNDED !OPP2! BOUNDED  → BOUNDED,
572:  21173                        !OPP1! BOUNDED  → BOUNDED]];
573:  21214          OPP1 ← /[OPERATOR: ABS, EXP, SQRT, -];
574:  21247          OPP2 ← /[OPERATOR: +, -, *, ↑];
```

APPENDIX 19

A PROCEDURE TO PROVE INEQUALITIES BY THE METHOD OF TRANSITIVITY

```
071:          SN        LINE 2          PROCEDURE TRANSPROOF
072:   12133 AL BOOLEAN PROCEDURE TRANSPROOF(F,G); VALUE F,G; FORM F,G;
073:   12152    BEGIN SYMBOL PH; FORM H;
074:   12161    SWITCH CASE ← CASE1,CASE2,NEXT,CASE4;

075:          SN        LINE 1          PROCEDURE BREAK
076:   12201    PROCEDURE BREAK(H); VALUE H; FORM H;
077:   12212    BEGIN IF H==P1 THEN BEGIN
078:   12222    C1: QA←A1; INSERT QA AFTER LAST OF PH; GOTO C2 END;
079:   12254    IF H==P2 THEN BEGIN
080:   12264    QA←A2; INSERT QA AFTER LAST OF PH; GOTO C1 END;
081:   12315    C2: END;

082:          SN        LINE 1          PROCEDURE HEURISTEST
083:   12317 AL INTEGER PROCEDURE HEURISTEST;
084:   12322    BEGIN INTEGER I,A,B; REAL EF,EG,EH;
085:             COMMENT:    HEURISTEST:
086:                              1              F > H > G
087:                              2              F > G > H  OR  H > F > G
088:                              3              NO RELATION
089:                              4              F < H < G  ;
090:   12325    FOR I←1 STEP 1 UNTIL 5 DO BEGIN
091:   12356    B←0;
092:   12362    PREP;
093:   12365    EF ← EVAL([X1])F([S1]);
094:   12404    EG ← EVAL([X1])G([S1]);
095:   12423    EH ← EVAL([X1])H([S1]);
096:   12442    B ← (IF EF > EH THEN 1 ELSE 2)*(IF EH > EG THEN 1 ELSE 3);
097:   12464    IF B=6 THEN BEGIN A←4; GOTO FINH END;
098:   12474    IF I=1 THEN A←B ELSE
099:   12503    IF B=A THEN BEGIN A←3; GOTO FINH END END;
100:   12516    IF A=3 THEN A←2;
101:   12524    FINH: HEURISTEST←A;
102:   12527    PRINT(,F=F^,H=H^,G=G^,A=A);
103:   12604    END;

104:          SN        LINE 1
105:   12607    TRANSPROOF←FALSE;
106:   12612    PH ← 1; A1 ← F > H:ANY; A2 ← H:ANY >G;
107:   12653    FOR QA←ELEMENTS OF GREAT DO BEGIN
108:   12677    AX←QA;  IF (AX==A1)∨(AX==A2) THEN BEGIN
109:   12722    QA←H; INSERT QA AFTER LAST OF PH END END;
110:   12753    BREAK(F); BREAK(G);
111:   12766    PRINT(PH);

112:   12772    NEXT: H ←1ST OF PH; DELETE 1ST OF PH;
113:   13032    PRINT(,H=H);
114:   13042    IF EMPTY(PH) THEN GOTO FINT; GOTO CASE(HEURISTEST);
115:   13057    CASE1: IF PROVECONJ(F>H) THEN BEGIN
116:   13100             IF PROVECONJ(H>G) THEN BEGIN
117:   13121          TRANSPROOF←TRUE; GOTO FINT END END; GOTO NEXT;
118:   13127    CASE2: BREAK(H); GOTO NEXT;
119:   13136    CASE4: PRINT( ,WRONG←,CONJETURE); ALARM←TRUE; GOTO FINT;
120:   13155    FINT: END;
```

APPENDIX 19 CONTINUED

```
040:          SN        LINE 2          PROCEDURE PREP
041:    11264           PROCEDURE PREP;
042:    11270           BEGIN INTEGER I,J,L1,L2; FORM L; SYMBOL H; REAL X;
043:    11277           I ← LAST OF <Y>;
044:    11324           FOR J←1 STEP 1 UNTIL I DO BEGIN
045:    11355           H←<J TH OF THE BOUNDS OF Y>;
046:    11404           L← J TH OF <Y>;
047:    11431           L1 ← 1 ST OF H;
048:    11454           L2 ← 2 ND OF H;
049:    11477           X ← RANDOM(L1,L2); IF J=1 THEN BEGIN
050:    11513           R1←L=X; S←X END ELSE BEGIN QA←X;
051:    11546           R1←R1^L=X; !NSERT QA AFTER LAST OF S END END;
052:    11606           END;

147:    13535           P1 ← !OP1! A1:ANY;
148:    13554           P2 ← A1:ANY !OP2! A2:ANY;
149:    13603           OP1←/[OPERATOR:-,SQRT,ABS,SIN,COS,LN,EXP][INDEX:J];
150:    13670           OP2←/[OPERATOR:+,-,*,/,↑][INDEX:J];
```

ALFORITHM FOR GENERATING CONJECTURES ABOUT INEQUALITIES

```
      SN        LINE 3          PROCEDURE FINDRELAT
 15134          PROCEDURE FINDRELAT(F,G,Y); VALUE F,G,Y;
 15137          FORM F,G; SYMBOL Y;
 15162          BEGIN INTEGER I; REAL T1,T;


 15165          PROCEDURE TRYONE(A); REAL A;
 15171          BEGIN INTEGER I,J,L1,L2; FORM R1,R,L; SYMBOL X1,H,S; REAL X,EF,EG,B;
  5210          I ← LAST OF <Y>;
  5235          FOR J←1 STEP 1 UNTIL I DO BEGIN
 15266          H←<J TH OF THE BOUNDS OF Y>;
  5315          L← J TH OF <Y>;
  5342          L1 ← 1 ST OF H;
 15365          L2 ← 2 ND OF H;
 15410          X ← RANDOM(L1,L2);  IF J=1 THEN BEGIN
  5423          R1←L=X; S←X END ELSE BEGIN X1←X;
  5454          R1←R1^L=X; INSERT X1 AFTER LAST OF S END END;
 15511          X1←<Y>; EF←EVAL([X1])F([S]); EG←EVAL([X1])G([S]);
  5551          B←EF-EG; IF ABS(B)<1.-6 THEN B←0;
  5564          B←SIGN(B); R← IF B=0 THEN .F=.G ELSE IF B=1 THEN .F>.G ELSE .F<.G;
 15637          PRINT(.WITH-R1-(SUBS(.F,.G)R(F,G)));

  5717          A←B; END PROCEDURE TRYONE;

      SN        LINE 3          BEGINS BODY OF PROCEDURE FINDRELAT
 15726          IF GREATER(F,G)>2 THEN BEGIN
 15740 SN       CMPL 1770005316
  5742          PRINT(.SOME-.NUMERIC-.INSIGHT);
  5764 SN       CMPL 1770005316
 15766          TRYONE(T1);
  5773          FOR I←1 STEP 1 UNTIL 5 DO BEGIN
  6024          TRYONE(T);
 16032          IF T*T1<0 THEN BEGIN
 16040          PRINT(.THUS-F>G^F<G); GOTO CA END;
  6067          IF T*T1>0 THEN GOTO CONTI;
  6076          IF T=T1 THEN GOTO CONTI;
 16103          T1 ← IF(IF T=0 THEN T1 ELSE T)>0 THEN 2 ELSE -2;
  6121          CONTI: END;

      SN        LINE 1
  6124          CONJECTURE ← SUBS(.A,.B)BASICFORM(15+T1](F,G);
  6201 SN       CMPL 1770005316
 16203          PRINT(CONJECTURE:CONJECTURE);
                CA: END;
  6214          END;
```

APPENDIX 20 CONTINUED

```
 17061          GREAT ← [.X=.X];
 17101          XRAND←23641;
 17104          YY←[.A,.B,2]; YY←/[BOUNDS:[-100.5,100.5],[-100.5,100.5]];

 17527     FINDRELAT( (.A+.B)/2, SQRT((.A↑2+.B↑2)/2), .YY);
 17621     YY ← /[BOUNDS: [0.250], [0.25]] ];
 17662     FINDRELAT( 2/(1/.B + 1/.A), SQRT(.B*.A), .YY);
 17747          END

BEGIN EXECUTION 14:40:35;  ⌾05017 AVAILABLE CELLS

SOME - NUMERIC - INSIGHT

WITH - (A=(.59899902344⌾+02)^B=(.97094726562⌾+02)) - ((A + B)/2<SQRT((A↑2 + B↑2)/2))
WITH - (A=(-.24285888672⌾+02)^B=(.14581298828⌾+02)) - ((A + B)/2<SQRT((A↑2 + B↑2)/2))
WITH - (A=(.35681152344⌾+02)^B=(-.34411621094⌾+02)) - ((A + B)/2<SQRT((A↑2 + B↑2)/2))
WITH - (A=(.50830078125⌾+02)^B=(-.87860546875⌾+02)) - ((A + B)/2<SQRT((A↑2 + B↑2)/2))
WITH - (A=(.35278320312⌾+02)^B=(.69952392578⌾+02)) - ((A + B)/2<SQRT((A↑2 + B↑2)/2))
WITH - (A=(.62591552734⌾+02)^B=(.24902343750⌾+02)) - ((A + B)/2<SQRT((A↑2 + B↑2)/2))

CONJECTURE:((A + B)/2<SQRT((A↑2 + B↑2)/2))

SOME - NUMERIC - INSIGHT

WITH - (A=(.22485351562⌾+03)^B=(.26895996328⌾+02)) - (2/(1/B + 1/A)<SQRT(B*A))
WITH - (A=(.90805053711⌾+02)^B=(.33248901367⌾+02)) - (2/(1/B + 1/A)<SQRT(B*A))
WITH - (A=(.19393920898⌾+03)^B=(.16946047559⌾+03)) - (2/(1/B + 1/A)<SQRT(B*A))
WITH - (A=(.96496582031⌾+02)^B=(.19331359863⌾+03)) - (2/(1/B + 1/A)<SQRT(B*A))
WITH - (A=(.55847167969⌾+01)^B=(.12339019775⌾+03)) - (2/(1/B + 1/A)<SQRT(B*A))
WITH - (A=(.19223785400⌾+03)^B=(.22438049316⌾+03)) - (2/(1/B + 1/A)<SQRT(B*A))

CONJECTURE:(2/(1/B + 1/A)<SQRT(B*A))
```

APPENDIX 21

A PROCEDURE TO PROVE INEQUALITIES BY THE METHOD OF ANALYSIS

```
365:         SN       PAGE PROCEDURE ANALPROOF;
366:   22156 AL BOOLEAN PROCEDURE ANALPROOF(R); FORM R;
367:            BEGIN
368:            COMMENT: 'ANALITIC PROOF', THIS PROCEDURE IS TRUE IF THE RELATION R
369:                     IS PROVED ANALITICALLY BY PROCEDURE GREATER;
370:   22161    BOOL←R==P20; ANALPROOF←GREATER(AX,BX)<4 END;


059:   11746            INTEGER PROCEDURE GREATER(F,G); VALUE F,G; FORM F,G;


060:         SN        LINE 3
061:                   COMMENT: IF FOR ALL VALUES OF PARAMETERS IN F AND G IT IS THE C
062:                   1.- F > G THEN GREATER =  1
063:                   2.- F¬< G THEN GREATER =  2
064:                   3.- F = G THEN GREATER =  0
065:                   4.- F < G THEN GREATER = -1
066:                   5.- F¬> G THEN GREATER = -2
067:                   6.- CAN'T TELL:GREATER = 10
068:
069:                   'GREAT' IS A LIST OF FORMULAE OF THE FORM 'A>B' OR 'A¬<B' WHICH
070:                   BY DEFINITION ARE TRUE INEQUALITIES;
071:
072:   11765           BEGIN INTEGER SF,SG,K,J1,J2,K1,K2,I,CASE,TEMP,REASON;
073:   11770           SWITCH OPER ← MAS,MENOS,POR,ENTRE,ALA,UNMENOS,RAIZ,
074:   12010                        ABSOL, TRIG, TRIG, LOGA, EXPO;
075:   12041           SWITCH CONTINUE←CASE1,CASE2,CASE3,CASE4,CASE5;
076:   12063           FORM AX,BX,CX,DX,RX,SX,TX;

077:         SN        LINE 1
078:   12102           BOOLEAN PROCEDURE JUST5(A,B,C,D); VALUE A,B,C,D; FORM A,B,C,D;
079:                   BEGIN
080:   12136           BOOLEAN PROCEDURE JAUX5(A,B,C,D); VALUE A,B,C,D; FORM A,B,C,D;
081:   12174           BEGIN INTEGER K1,K2,K3;
082:   12177           JAUX5←FALSE;
083:   12202           K1 ← GREATER(A,C); IF K1>3 THEN GOTO EL5;
084:   12221           K2 ← GREATER(B,D); IF K2>3 THEN GOTO EL5;
085:   12240           K3 ← K2*K1; IF K3<0 THEN GOTO EL5;
086:   12252           IF K1=K2 THEN BEGIN TEMP←K1; GOTO GOOD5 END;
087:   12263           TEMP ← IF K3=0 THEN (
088:   12267                  IF K1=0 THEN K2 ELSE K1) ELSE 1;
    N
089:   12302           GOOD5: JAUX5←TRUE; EL5: END PROCEDURE JAUX5;
090:   12307           JUST5←TRUE;
091:   12312           IF JAUX5(A,B,C,D) THEN GOTO EJ5;
092:   12331           IF JAUX5(A,B,D,C) THEN GOTO EJ5;
093:   12350           JUST5←FALSE; EJ5: REASON←5 END;
```

```
094:          SN        LINE 1
095:  12360             BOOLEAN PROCEDURE JUST7(A,B,C,D); VALUE A,B,C,D; FORM A,B,C,D;
096:                    BEGIN
097:  12413             BOOLEAN PROCEDURE JAUX7(A,B,C,D); VALUE A,B,C,D; FORM A,B,C,D;
098:  12451             BEGIN INTEGER K1,K2,K3,AS,BS,CS,DS;
099:  12454             JAUX7←FALSE;
100:  12457             K1 ← GREATER(A,C); IF K1>3 THEN GOTO EX7;
101:  12476             K2 ← GREATER(B,D); IF K2>3 THEN GOTO EX7;
102:  12515             K3 ← K1*K2; IF K3<0 THEN GOTO EX7;
103:  12527             AS←TESTSIGN(A); BS←TESTSIGN(B); CS←TESTSIGN(C); DS←TESTSIGN(D);
104:  12564             IF K3>0 THEN BEGIN
105:  12571             IF K1>0 THEN BEGIN
106:  12576             IF CS¬=1 THEN BEGIN
107:  12603             IF AS¬=-1 THEN GOTO EX7; IF BS¬=-1 THEN GOTO EX7;
108:  12620             P71: TEMP←IF K3<3 THEN -1 ELSE -2; GOTO GOOD7 END;
109:  12634             IF DS¬=1 THEN GOTO EX7;
110:  12643             P72: TEMP←IF K3<3 THEN 1 ELSE 2; GOTO GOOD7 END;
111:  12657             IF AS¬=1 THEN BEGIN
112:  12664             IF CS¬=-1 THEN GOTO EX7; IF DS¬=-1 THEN GOTO EX7; GOTO P72 END;
113:  12702             IF BS¬=1 THEN GOTO EX7; GOTO P71 END;
114:  12712             IF K1=K3 THEN BEGIN TEMP←3; GOTO GOOD7 END;
115:  12723             IF K1=0 THEN BEGIN
116:  12727             IF AS¬=0 THEN BEGIN K3←AS;
117:  12736             P73: TEMP←K2*K3; GOTO GOOD7 END;
118:  12745             IF CS=0 THEN GOTO EX7; K3←CS; GOTO P73 END;
119:  12756             IF BS¬=0 THEN BEGIN K3←BS;
120:  12765             P74: TEMP←K1*K3; GOTO GOOD7 END;
121:  12774             IF DS=0 THEN GOTO EX7; K3←DS; GOTO P74;
122:  13005             GOOD7: JAUX7←TRUE;
      N
123:  13010             EX7: END PROCEDURE JAUX7;
124:
125:  13013             JUST7←TRUE;
126:  13016             IF JAUX7(A,B,C,D) THEN GOTO EJ7;
127:  13035             IF JAUX7(A,B,D,C) THEN GOTO EJ7;
128:  13054             IF JAUX7(-A,-B,D,C) THEN GOTO EJ7;
129:  13105             IF JAUX7(-A,-B,C,D) THEN GOTO EJ7;
130:  13136             IF JAUX7(A,B,-C,-D) THEN GOTO EJ7;
131:  13167             IF JAUX7(A,B,-D,-C) THEN GOTO EJ7;
132:  13220             IF JAUX7(-A,-B,-C,-D) THEN GOTO EJ7;
133:  13263             IF JAUX7(-A,-B,-D,-C) THEN GOTO EJ7;
134:  13326             JUST7←FALSE; EJ7:REASON←7 END;
```

```
135:          SN        LINE 2          BEGINS BODY OF PROCEDURE GREATER

136:          SN        LINE 2
137:    13336           REASON ← 2;
138:    13341           IF (F==OF(NUM))∧(G==OF(NUM)) THEN BEGIN
139:    13361           AX ← F>G; GOTO IF EVAL AX THEN UNO ELSE CUATRO END;
140:    13403           IF F>G == OF(GREAT) THEN BEGIN
141:    13422           UNO: TEMP←1; GOTO EXITG END;
142:    13427           IF F¬<G == OF(GREAT) THEN BEGIN
143:    13446           DOS: TEMP←2; GOTO EXITG END;
144:    13453           IF G>F == OF(GREAT) THEN BEGIN
145:    13472           CUATRO: TEMP←-1; GOTO EXITG END;
146:    13477           IF G¬<F == OF(GREAT) THEN BEGIN
147:    13516           CINCO:  TEMP←-2; GOTO EXITG END;
148:    13523           SF ← TESTSIGN(F); SG ← TESTSIGN(G);
149:    13542           IF SF=SG THEN BEGIN
150:    13546           IF F==G THEN GOTO TRES; K←SF; GOTO GR1 END;
151:    13564           REASON←1;
152:    13567           IF SF=0 THEN BEGIN
153:    13573           IF F==0 THEN BEGIN TEMP←-SG; GOTO EXITG END; K←SG*2; GOTO GR1 E
154:    13615           IF SG=0 THEN BEGIN                                          END
155:    13621           IF G==0 THEN GOTO GR0; K←SF*3; GOTO GR1 END;
156:    13641           GR0: TEMP ← SF; GOTO EXITG ;
157:                    GR1: COMMENT:
158:                                        G>0       G<0       G ¬KNOWN
159:                              F>0       K=1       SOLVED    K=3
160:                              F<0       SOLVED    K=-1      K=-3
161:                              F ¬KNOWN  K=2       K=-2      K=0;
163:    13646           J1←J2←0;
164:    13652           IF F==P1 THEN BEGIN AX←A1;J1←J+5 END;
165:    13667           IF F==P2 THEN BEGIN AX←A1; BX←A2; J1←J END;
166:    13705           IF G==P1 THEN BEGIN CX←A1; J2←J+5 END;
167:    13722           IF G==P2 THEN BEGIN CX←A1; DX←A2; J2←J END;

168:          SN        LINE 2
169:    13740           IF (J1=1 ∨ J1=2)∧(J2=1 ∨ J2=2) THEN BEGIN
170:                    COMMENT:    GREATER( AX1+,-1BX, CX1+,-1DX);
171:    13772           IF J1=2 THEN BX←-BX;
172:    14002           IF J2=2 THEN DX←-DX;  J1←J2←1;
173:    14015           IF JUST5(AX,BX,CX,DX) THEN GOTO EXITG;
174:    14034           GOTO ONEATIME END;

175:          SN        LINE 1
176:    14037           IF (J1=3 ∨ (J1=4 ∧ ¬(AX==1))) ∧
177:    14061              (J2=3 ∨ (J2=4 ∧ ¬(CX==1))) THEN BEGIN
178:    14110           IF J1=4 THEN BX←SIMPLIFY(1/BX);
179:    14133           IF J2=4 THEN DX←SIMPLIFY(1/DX);
180:    14156           IF JUST7(AX,BX,CX,DX) THEN GOTO EXITG;
181:    14175           GOTO ONEATIME END;

182:          SN        LINE 1
183:    14200           IF (J1=9 ∨ J1=10)∧(J2=9 ∨ J2=10) THEN
184:    14232           COMMENT   GREATER( ISIN,COSI , ISIN,COSI);  GOTO SIETE;
```

```
186:  14235           IF J1=J2 THEN BEGIN
187:  14241           IF J1=0 THEN GOTO SEIS;
188:  14247           IF J1=6 THEN BEGIN
189:  14253           TEMP←GREATER(CX,AX); REASON←4; GOTO EXITG END;
190:  14270           IF J1=7 ∨ J1=11 ∨ J1=15 THEN BEGIN
191:                  MONOINC: COMMENT, MONOTONIC INCREASING FUNCTIONS PRESERVE THE
192:                  BETWEEN ITS ARGUMENTS;
193:  14314           TEMP←GREATER(AX,CX); REASON←3; GOTO EXITG END END;

194:        SN        LINE 1
195:                  ONEATIME:
196:  14331           IF J2>0 THEN BEGIN CASE←1;
197:  14340           I←1;  TX←F; RX←CX; SX←DX; GOTO OPER[J2] END;
198:  14353           CASE1: IF J1>0 THEN BEGIN CASE←2;
199:  14362           I←-1;TX←G; RX←AX; SX←BX; GOTO OPER[J1] END;

200:        SN        LINE 2
201:                  CASE2: COMMENT,
202:                  THE ANALYTIC APPROACH HAS FAILED, LET'S TRY SINTHETIC METHODS;
203:  14375           GOTO SEIS;

204:        SN        LINE 2
205:  14400           TRES: TEMP←0; GOTO EXIT3;
206:  14405           SEIS: TEMP←10;GOTO EXIT3;

207:        SN        LINE 2
208:                  MAS:  COMMENT      GREATER(RX+SX, TX)       IF I=-1
209:                                     GREATER(TX, RX+SX)       IF I= 1;
210:  14412           IF JUST5(0,TX,RX,SX) THEN BEGIN
211:  14427           TEMP←TEMP*I; GOTO EXITG END; GOTO CONTINUE[CASE];

212:        SN        LINE 2
213:                  MENOS: COMMENT     GREATER(RX-SX, TX)  IF I=-1
214:                                     GREATER(TX, RX-SX)  IF I= 1;
215:  14437           SX←-SX; GOTO MAS;

216:        SN        LINE 2
217:                  POR:    COMMENT    GREATER(RX*SX, TX)      IF I=-1
218:                                     GREATER(TX, RX*SX)      IF I= 1;
219:  14445           IF JUST7(1,TX,RX,SX) THEN BEGIN
220:  14462           TEMP ← TEMP*I; GOTO EXITG END; GOTO CONTINUE[CASE];

221:        SN        LINE 2
222:                  ENTRE:   COMMENT   GREATER( RX/SX, TX)  IF I=-1
223:                                     GREATER( TX, RX/SX)  IF I= 1;
224:  14472           K1← TESTSIGN(SX); IF K1=0 THEN GOTO CONTINUE[CASE];
225:  14507           IF K1>0 THEN BEGIN
226:  14514           ENTRE1:  IF JUST7(SX,TX,1,RX) THEN BEGIN
227:  14531           TEMP←TEMP*I; REASON←9;GOTO EXITG END; GOTO CONTINUE[CASE] END;
228:  14543           SX←-SX; RX←-RX; GOTO ENTRE1;
```

```
229:          SN        LINE 2
230:                    ALA: COMMENT        GREATER(RX↑SX, TX)  IF I=-1
231:                                        GREATER(TX, RX↑SX)  IF I= 1;
232:  14555             GOTO CONTINUE[CASE];

233:          SN        LINE 2
234:                    UNMENOS: COMMENT    GREAT(-RX, TX) IF I=-1
235:                                        GREATER(TX, -RX) IF I= 1;
236:  14560             GOTO CONTINUE[CASE];

237:          SN        LINE 2
238:                    RAIZ: COMMENT       GREATER( SQRT(RX), TX )    IF I=-1
239:                                        GREATER( TX, SQRT(RX) )    IF I= 1;
240:  14563             IF 1 = (IF I=1 THEN SF ELSE SG) THEN BEGIN
241:  14574             K1 ← GREATER(RX,TX↑2);
242:  14615             TEMP← IF K1<3 THEN -I*K1 ELSE K1; GOTO EXITG END;
243:  14631             GOTO CONTINUE[CASE];

244:          SN        LINE 2
245:                    ABSOL:  COMMENT     GREATER( ABS(RX), TX )   IF I=-1
246:                                        GREATER( TX, ABS(RX) )   IF I= 1;
247:  14634             K1←TESTSIGN(RX);
248:  14644             IF K1¬=0 THEN BEGIN
249:  14651             K2 ← GREATER(K1*RX,TX);
250:  14672             TEMP← IF K2<3 THEN -I*K2 ELSE K2; GOTO EXITG END;
251:  14706             GOTO CONTINUE[CASE];

252:          SN        LINE 2
253:                    TRIG:    COMMENT    GREATER(ISIN,COSI(RX), TX)  IF I=-1
254:                                        GREATER(TX, ISIN,COSI(RX))  IF I= 1;
255:  14711             K1 ← GREATER(TX,1);
256:  14722             IF K1=0 THEN BEGIN TEMP←2*I; GOTO EXITG END;
257:  14733             IF K1>0 THEN BEGIN
258:  14740             IF K1<3 THEN BEGIN
259:  14745             TRIG1: TEMP←K1*I; GOTO EXITG END END; K2←K1;
260:  14755             SX←-1; K1←GREATER(TX,SX);
261:  14771             IF K1=0 THEN BEGIN TEMP←-2*I; GOTO EXITG END;
262:  15002             IF K1<0 THEN GOTO TRIG1;
263:  15010             IF K1=10∧K2=10 THEN GOTO CONTINUE[CASE]; GOTO SIETE;
```

```
             SN          LINE 2
                         LOGA: COMMENT          GREATER( LN(RX), TX )  IF I=-1
                                                GREATER( TX, LN(RX) )  IF I= 1;
     15032               K1 ← GREATER(TX,AX);
     15043               IF K1>-1∧K1<3 THEN BEGIN TEMP←I; GOTO EXITG END;
     15065               GOTO CONTINUE[CASE];

             SN          LINE 2
                         EXPO:  COMMENT         GREATER( EXP(RX), TX )  IF I=-1
                                                GREATER( TX, EXP(RX) )  IF I= 1;
     15070               K1 ← GREATER(RX,TX);
     15101               IF K1>-1∧K1<3 THEN BEGIN TEMP←-I; GOTO EXITG END;
     15123               IF TESTSIGN(TX)=1 THEN BEGIN
     15134               K1 ← GREATER(RX,LN(TX));
     15152               IF K1<3 THEN BEGIN TEMP←I*K1; GOTO EXITG END END;
     15164               GOTO CONTINUE[CASE];

             SN          LINE 1
     15167               EXITG:  GREATER←TEMP;
     15172               IF TEMP<4 ∧ REASON¬=2 THEN BEGIN
     15210               AX ← SUBS(.A,.B)BASICFORM[TEMP+15](F,G);
     15266               QA←AX; INSERT QA AFTER LAST OF GREAT;
     15315               PRINT(AX-.BY-(.REASON=REASON)) END;
     15341               END PROCEDURE GREATER;

             SN          PAGE BODY OF MAIN PROGRAM

     16436                P1 ← [OP1] A1:ANY;
     16455                P2 ← A1:ANY [OP2] A2:ANY;
     16504                OP1←/[OPERATOR:-,SQRT,ABS,SIN,COS,LN,EXP][INDEX:J];
     16571                OP2←/[OPERATOR:+,-,*,/,↑][INDEX:J];
     16644                NUM←[REAL,INTEGER];
     16661                BASICFORM[1] ← .A+.B;
     16703                BASICFORM[2] ← .A-.B;
     16725                BASICFORM[3] ← .A*.B;
     16747                BASICFORM[4] ← .A/.B;
     16771                BASICFORM[5] ← .A↑.B;
     17013                BASICFORM[6] ← -.A;
     17030                BASICFORM[7] ← SQRT(.A);
     17045                BASICFORM[8] ← ABS(.A);
     17062                BASICFORM[9] ← SIN(.A);
     17077                BASICFORM[10]← COS(.A);
     17114                BASICFORM[11]← LN(.A);
     17131                BASICFORM[12]← EXP(.A);
     17146                BASICFORM[13]← .A ¬> .B;
     17170                BASICFORM[14]← .A  < .B;
     17212                BASICFORM[15]← .A  = .B;
     17234                BASICFORM[16]← .A  > .B;
     17256                BASICFORM[17]← .A ¬< .B;

     17300                GREAT ← [.X=.X];
     17320                XRAND←23641;
```

APPENDIX 22

A PROCEDURE TO PROVE INEQUALITIES BY THE METHOD OF LOGICAL CHAIN (CHAINPROOF) AND ITS DRIVER (PROVECONJ).

```
378:          SN        LINE 2          PROCEDURE CHAINPROOF
379:    22263       BOOLEAN PROCEDURE CHAINPROOF(F,IFF,SIMPL); VALUE IFF,SIMPL;
380:    22266       SYMBOL IFF,SIMPL; FORM F;
381:    22303       BEGIN INTEGER I,J; FORM R;
382:    22310       R←F; CHAINPROOF←TRUE;
383:    22317       IFF←/[INDEX:I]; SIMPL←/[INDEX:J];
384:    22350       L1: R←R.↓IFF; IF I=2 THEN BEGIN L2:
385:    22361   SN          CMPL 1770005316
386:    22363   AL  PRINT(R); IF CELLS<500 THEN GO;
387:    22373       IF ANALPROOF(R) THEN GOTO FIN; GOTO L1 END;
388:    22410       R ← R.↓SIMPL; IF J=2 THEN GOTO L2;
389:    22422       F←R; CHAINPROOF←FALSE; FIN:END;


390:          SN        LINE 3          PROCEDURE PROVECONJ
391:    22433   AL  PROCEDURE PROVECONJ(R); VALUE R; FORM R;
392:    22444       BEGIN SYMBOL NOSIMP; FORM F;
393:    22453   SN          CMPL 1770005316
394:    22455       PRINT(R);
395:    22460       GO;
396:    22462       IF CHAINPROOF(R,IFF1,NOSIMP) THEN GOTO DONE;
397:    22512       BOOL←R==P20; R←0<RX←AX;
398:    22533       IF CHAINPROOF(R,IFF2,MINUSOUT) THEN GOTO DONE;
399:    22563       IF CHAINPROOF(R,IFF2,RATIONAL) THEN GOTO DONE;
400:    22613       IF CHAINPROOF(R,IFF2,EXPAND) THEN GOTO DONE;
401:    22643       PASS1←1; PASS2←0;
402:    22650       BOOL←R==P20; IF AX==0 THEN BEGIN F←0<.YY; AX←RX END ELSE F←.YY<0;
403:    22710       IF AX==P19 THEN BEGIN YY←.YY/.PB; ZZ←REPLACE(F); D←POLY(PA);
404:    22743       YY←.D/.YY; ZZ←REPLACE(F); YY←D/POLY(PB) END ELSE
405:    22775       BEGIN ZZ←F; YY←POLY(AX) END; R←REPLACE(F);
406:    23012   SN          CMPL 1770005316
407:    23014       PRINT(R);
408:    23017       IF CHAINPROOF(R,IFF2,NOSIMP) THEN GOTO DONE;
409:    23047       BOOL ← R==P20;  IF FACTOR(BX,YY) THEN BEGIN
410:    23067       R ← REPLACE(F);
411:    23073   SN          CMPL 1770005316
412:    23075       PRINT(R);
413:    23100       IF CHAINPROOF(R, IFF2, NOSIMP) THEN GOTO DONE END;
414:    23130       DONE: END;
```

APPENDIX 23

SOME OUTPUT OF INEQUALITY'S PROOFS

```
526:  30724 AL F ← (.A+.B)/2 < SQRT( (.A↑2 + .B↑2)/2 );
527:  31002    PROVECONJ(F);
528:  31007    F ← 2/(1/.B + 1/.A) < SQRT(.B*.A);
529:  31060 SN       CMPL 1770005324
530:  31061    PROVECONJ(F);
531:  31066          END
```

```
BEGIN EXECUTION 00:33:00;  02568 AVAILABLE CELLS

(A + B)/2 < SQRT((A↑2 + B↑2)/2)
A + B>0
(A + B)/2>0

((A + B)/2)↑2 < (A↑2 + B↑2)/2
((A + B)/2)↑2>0

2*((A + B)/2)↑2 < A↑2 + B↑2
(2*((A + B)/2)↑2>0)

0 < A↑2 + B↑2 - 2*(A + B)↑2/4

0 < (2*(A↑2 + B↑2) - (A + B)↑2)/2

0 < 2*(A↑2 + B↑2) - (A + B)↑2

0 < 2*(A↑2 + B↑2) - (A↑2 + B↑2 + 2*A*B)

0 < 2*A↑2 + 2*B↑2 - (A↑2 + B↑2 + 2*A*B)

0 < 2*A↑2 + 2*B↑2 - (A↑2 + B↑2 + 2*A*B)

0 < 2*A↑2 + 2*B↑2 - (A↑2 + B↑2 + 2*A*B)

0 < A↑2 + B↑2 - 2*A*B

0 < (B - A)↑2
```

APPENDIX 23 CONTINUED

```
2/(1/B + 1/A) < SQRT(B*A)
1/B>0
1/A>0
1/B + 1/A>0
2/(1/B + 1/A)>0

(2/(1/B + 1/A))↑2 < 3*A
(2/(1/B + 1/A))↑2>0

0 < B*A - 4/(1/B + 1/A)↑2

0 < (B*A*(1/B + 1/A)↑2 - 4)/(1/B + 1/A)↑2
(1/B + 1/A)↑2>0

0 < B*A*(1/B + 1/A)↑2 - 4

0 < B*A*((A + B)/(B*A))↑2 - 4

0 < B*A*(A + B)↑2/(B*A)↑2 - 4

0 < ((A + B)↑2*B*A - 4*(B*A)↑2)/(B*A)↑2
(B*A>0)
((B*A)↑2>0)

0 < (A + B)↑2*B*A - 4*(B*A)↑2

0 < (A + B)↑2*B*A - 4*B↑2*A↑2

0 < (A↑2 + B↑2 + 2*A*B)*B*A - 4*B↑2*A↑2

0 < (A↑2 + B↑2)*B*A + 2*A*B*B*A - 4*B↑2*A↑2

0 < A↑2*B*A + B↑2*B*A + 2*A*B*B*A - 4*B↑2*A↑2

0 < A↑2*B*A + B↑2*B*A + 2*A*B*B*A - 4*A↑2*B↑2

0 < A↑3*B + A*B↑3 + 2*A↑2*B↑2 - 4*A↑2*B↑2

0 < A↑3*B + A*B↑3 - 2*A↑2*B↑2

0 < A*B*(A - B)↑2
A*B>0

0 < (A - B)↑2
```

SOME OUTPUT OF LIMITING PROCESSES

```
TIME=(73502)^CELL=3590
F=(X↑2 + 2*X - 3)/(2*X↑2 + 2*X - 12)
(LIMIT,(F))=(.5000000000₁₀+00)
X→INFIP
DIREC=0

TIME=(73505)^CELL=2775
```

```
TIME=(73509)^CELL=3588
F=(X↑2 + 2*X - 3)/(2*X↑2 + 2*X - 12)
(LIMIT,(F))=(.4000000000₁₀+00)
X→-3
DIREC=0

TIME=(73527)^CELL=1092
```

```
TIME=(77089)^CELL=3627
F=1/H*((X + H)/(X + H + 1) - X/(X + 1))
ASSUME:((X*(X + 1) - X*(X + 1))/((X + 1)*(X + 1)))≠0
(X + 1)*(X + 1)>0
(LIMIT,(F))=SIGN((X*(X + 1) - X*(X + 1))/((X + 1)*(X + 1)))*INFIP
H→0
DIREC=1

TIME=(77097)^CELL=2629
```

```
TIME=(41125)^CELL=3658

F=N↑(1/N)

(LIMIT.(F))=1
N→INFIP

DIREC=0

TIME=(41129)^CELL=3285
```

```
TIME=(41171)^CELL=3671

F=(1 + A*H)↑(1/H)

ASSUME:A≠INFI
ASSUME:A≠INFI

(LIMIT.(F))=EXP(A)
H→0

DIREC=1

TIME=(41175)^CELL=2975
```

```
TIME=(73527)^CELL=1067

F=1/H*((X + H)/(X + H + 1) - X/(X + 1))

(LIMIT.(F))=H/((H + 1)*H)
X→0

DIREC=0

TIME=(73538)^CELL=2626
```

```
TIME=(73483)^CELL=2949

F=SIN(X)/X

(LIMIT.(F))=1
X→0

DIREC=0

TIME=(73491)^CELL=2328
```

```
TIME=(41130)^CELL=3675

F=LN(X)/SQRT(X)

(LIMIT.(F))=ZEROP
X→INFIP

DIREC=0

TIME=(41169)^CELL=3143
```

```
TIME=(73492)^CELL=2309

F=Y↑2*LN(X)

Y↑2>0
ASSUME:(Y↑2)≠0

(LIMIT.(F))=INFIM
X→0

DIREC=1

TIME=(73495)^CELL=2090
```

```
TIME=(73495)^CELL=2075

F=(X↑2 - 1)/(X - 1)

(LIMIT.(F))=2
X→1

DIREC=-1

TIME=(73499)^CELL=1556
```

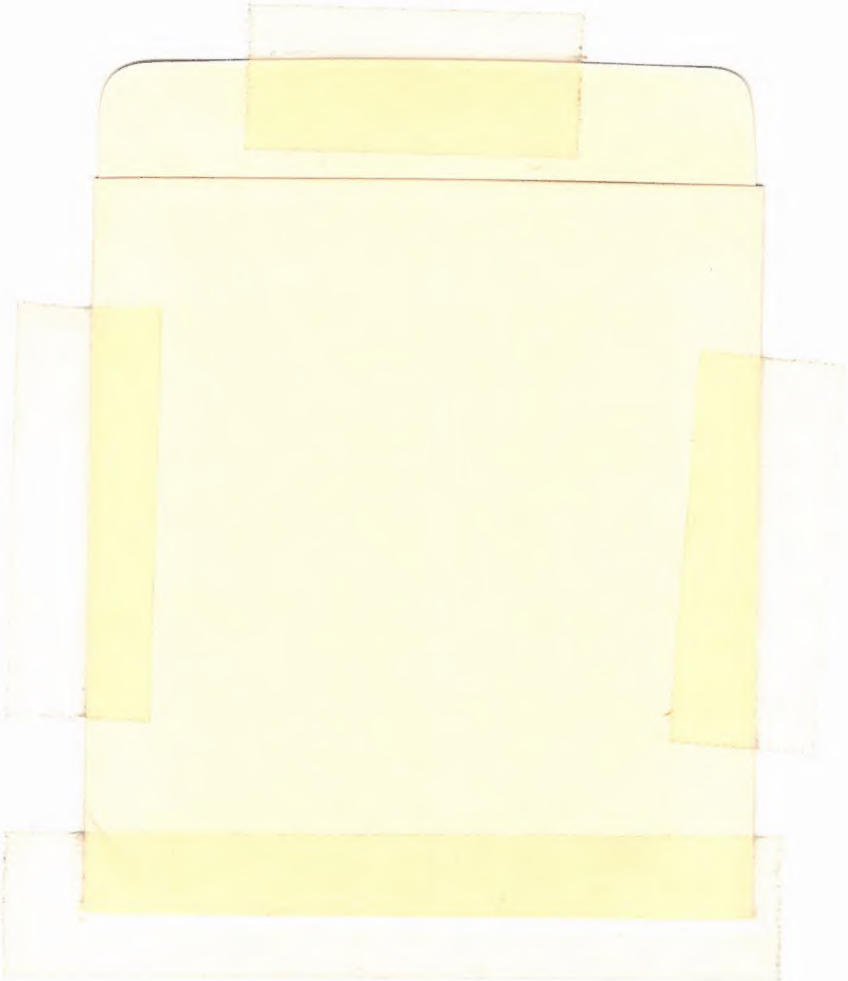
```
TIME=(73479)^CELL=3589

F=(X + 1)/(X - 1)

(LIMIT.(F))=UNDEF
X→1

DIREC=0

TIME=(73483)^CELL=2967
```

ASYMPTOTIC ANALYSIS EXAMPLE (C.F. APPENDIX 17)

```
F=1/H*((X + H)/(X + H + 1) - X/(X + 1))
H=1/Z
F=1/(1/Z)*((X + 1/Z)/(X + 1/Z + 1) - X/(X + 1))
(ORD.(1))=1*Z↑0
(ORD.(1))=1*Z↑0
(ORD.(Z))=1*Z↑1
(ORD.(1/Z))=1*Z↑-1
(ORD.(1/(1/Z)))=1*Z↑1
(ORD.(X))=X*Z↑0
(ORD.(1))=1*Z↑0
(ORD.(Z))=1*Z↑1
(ORD.(1/Z))=1*Z↑-1
(ORD.(X + 1/Z))=X*Z↑0
(ORD.(X))=X*Z↑0
(ORD.(1))=1*Z↑0
(ORD.(Z))=1*Z↑1
(ORD.(1/Z))=1*Z↑-1
(ORD.(X + 1/Z))=X*Z↑0
(ORD.(1))=1*Z↑0
X + 1
1 + X
(ORD.(X + 1/Z + 1))=(1 + X)*Z↑0
(ORD.((X + 1/Z)/(X + 1/Z + 1)))=X/(1 + X)*Z↑0
(ORD.(X/(X + 1)))=X/(X + 1)*Z↑0
X/(1 + X) + -(X/(X + 1))
0
(X + 1/Z)/(X + 1/Z + 1) + -(X/(X + 1))
Z↑2/(Z↑2 + Z↑3 + 2*X*Z↑3 + X*Z↑2 + X↑2*Z↑3)
(ORD.(Z))=1*Z↑1
(ORD.(Z↑2))=1*Z↑2
(ORD.(Z))=1*Z↑1
(ORD.(Z↑2))=1*Z↑2
(ORD.(Z))=1*Z↑1
(ORD.(Z↑3))=1*Z↑3
(ORD.(2))=2*Z↑0
(ORD.(X))=X*Z↑0
(ORD.(Z))=1*Z↑1
(ORD.(Z↑3))=1*Z↑3
(ORD.(X*Z↑3))=X*Z↑3
(ORD.(2*X*Z↑3))=2*X*Z↑3
(ORD.(X))=X*Z↑0
(ORD.(Z))=1*Z↑1
(ORD.(Z↑2))=1*Z↑2
(ORD.(X*Z↑2))=X*Z↑2
(ORD.(X↑2))=X↑2*Z↑0
(ORD.(Z))=1*Z↑1
(ORD.(Z↑3))=1*Z↑3
(ORD.(X↑2*Z↑3))=X↑2*Z↑3
(ORD.(X*Z↑2 + X↑2*Z↑3))=X↑2*Z↑3
2*X + X↑2
2*X + X↑2
(ORD.(2*X*Z↑3 + X*Z↑2 + X↑2*Z↑3))=(2*X + X↑2)*Z↑3
1 + 2*X + X↑2
1 + 2*X + X↑2
(ORD.(Z↑3 + 2*X*Z↑3 + X*Z↑2 + X↑2*Z↑3))=(1 + 2*X + X↑2)*Z↑3
(ORD.(Z↑2 + Z↑3 + 2*X*Z↑3 + X*Z↑2 + X↑2*Z↑3))=(1 + 2*X + X↑2)*Z↑3
(ORD.(Z↑2/(Z↑2 + Z↑3 + 2*X*Z↑3 + X*Z↑2 + X↑2*Z↑3)))=1/(1 + 2*X + X↑2)*Z↑-1
(ORD.((X + 1/Z)/(X + 1/Z + 1) - X/(X + 1)))=1/(1 + 2*X + X↑2)*Z↑-1
(ORD.(1/(1/Z)*((X + 1/Z)/(X + 1/Z + 1) - X/(X + 1))))=1/(1 + 2*X + X↑2)*Z↑0
```